NIIT 国家软件与集成电路公共服务平台信息技术紧缺人才培养工程指定教材

教材+教案+授课资源+考试系统+题库+教学辅助案例

一站式IT系列就业应用课程

MySQL数据库原理、设计与应用

黑马程序员 / 编著



- 本书基于MySQL5.7编写，涵盖了数据库基础知识、数据库优化和部署、大型项目数据库设计等内容，提供11个综合案例。
- 提供免费教学资源，包括12个精美教学PPT、900道测试题、长达20小时的教学视频等。

清华大学出版社

MySQL 数据库原理、设计与应用

黑马程序员　编著

清华大学出版社

北　京

内 容 简 介

本书是面向 MySQL 数据库初学者的入门教材，以通俗易懂的语言、丰富实用的案例，详细讲解 MySQL 的开发和管理技术。

全书共 12 章。第 1 章讲解数据库基本概念和 MySQL 的安装步骤；第 2、3 章讲解数据库的基本操作；第 4 章讲解数据库设计的理论与实践；第 5、6 章讲解单表和多表的查询操作；第 7～11 章讲解用户与权限、视图、事务、存储过程、索引等，适合需要提高自身技术的读者；第 12 章讲解 Linux 环境下 MySQL 的配置和部署方案。

本书附有配套资源，包括教学视频、习题、教学课件等，而且为了帮助读者更好地学习本书中的内容，还提供了在线答疑，希望得到更多读者的关注。

本书既可作为高等院校计算机相关专业的数据库基础课程的教材，也可作为广大 IT 技术人员和编程爱好者的读物。

图书在版编目(CIP)数据

MySQL 数据库原理、设计与应用/黑马程序员编著. —北京：清华大学出版社，2019
ISBN 978-7-302-52376-5

Ⅰ. ①M…　Ⅱ. ①黑…　Ⅲ. ①SQL 语言－程序设计　Ⅳ. ①TP311.132.3

中国版本图书馆 CIP 数据核字(2019)第 037362 号

责任编辑：袁勤勇　杨　枫
封面设计：刘新新
责任校对：李建庄
责任印制：李红英

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：清华大学印刷厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：23.25　　**字　　数**：564 千字
版　　次：2019 年 4 月第 1 版　　**印　　次**：2019 年 4 月第1 次印刷
定　　价：59.00 元

产品编号：081929-01

序

传智播客和"黑马程序员"

江苏传智播客教育科技股份有限公司(简称传智播客)是一家专门致力于高素质软件开发人才的科技公司。"黑马程序员"是传智播客旗下的高端IT教育品牌。

"黑马程序员"的学员多为大学毕业后想从事IT行业,但各方面条件还不成熟的年轻人。"黑马程序员"的学员筛选制度非常严格,包括严格的技术测试、自学能力测试,还包括性格测试、压力测试、品德测试等,百里挑一的残酷筛选制度确保学员质量,并降低企业的用人风险。

自"黑马程序员"成立以来,教学研发团队一直致力于打造精品课程资源,不断在产、学、研3个层面创新自己的执教理念与教学方针,并集中"黑马程序员"的优势力量,有针对性地出版了计算机系列教材60多种,制作了教学视频数十套,发表各类技术文章数百篇。

"黑马程序员"不仅斥资研发IT系列教材,还为高校师生提供以下配套学习资源与服务。

为大学生提供的配套服务:

1. 专注辅学的"高校学习平台"(http://yx.boxuegu.com),专业老师在线为您答疑解惑。

2. 针对高校学生在学习过程中存在的压力等问题,面向大学生量身打造了"播妞"。同学可以添加"播妞"微信:208695827/ QQ:3231342131,获取学习资源。



微信二维码



QQ二维码

3. 高校学生也可扫描上方右侧二维码,加入播妞粉丝团,获取最新学习资源,与播妞一起快乐学习。

为IT教师提供的配套服务:

针对高校教学,"黑马程序员"为IT系列教材精心设计了"教案+授课资源+考试系统

＋题库＋教学辅助案例”等系列教学资源。高校老师请登录“高校教辅平台”(http://yx.boxuegu.com)平台或关注码大牛老师微信/QQ：2011168841，获取教材配套资源，也可以扫描下方二维码，加入专为IT教师打造的师资服务平台——“教学好助手”，获取“黑马程序员”最新教师教学辅助资源的相关动态。

前 言

MySQL 是一种关系数据库管理系统，它是目前世界上流行的数据库之一，具有开源、稳定、可靠、管理方便以及支持众多系统平台等特点。MySQL 广泛应用于互联网行业的数据存储，如电商、社交等网站数据的存储往往都是 MySQL。

目前，从各大招聘网站的信息来看，各类计算机人才的技能要求中基本都要掌握至少一种数据库的操作和使用。其中，MySQL 数据库是最常见的一种。因此，MySQL 数据库一般会作为计算机相关专业需要了解或掌握的技能之一。

为什么要学习本书

本书针对想要从事与计算机相关的工作，但是没有数据库基础或基础比较薄弱的人群。从了解数据库的特点、概念、原理起步，再探讨 MySQL 数据库的特点和使用，尽可能地确保读者可以学以致用，具备解决实际问题的能力。

本书根据知识的难易程度，采用先易后难的方式部署教材章节顺序。在知识讲解时以环环相扣的推进方式阐述出每个名词概念的作用以及相互之间的联系；在实际操作时，从指令的语法、注意事项、案例演示等多个角度详细讲解，帮助读者提高对 MySQL 数据库的整体认识。

如何使用本书

本书主要讲解的内容包括数据库的理论知识，MySQL 数据库安装与配置，数据库、数据表的管理，数据的增、删、改、查操作，以及用户权限、视图、存储引擎、事务、索引等，还增加了电子商务网站的数据库设计以及 Linux 环境的配置与部署。

本书共分为 12 章，各章内容简要介绍如下。

- 第 1 章主要从数据库系统的组成、数据库技术的发展阶段、数据库的三级模式结构、数据模型与关系运算、SQL 等方面介绍与数据库相关的理论。还讲解了 MySQL 安装与配置、常用图形化工具的使用。通过本章的学习，要求初学者对数据库在理论体系上有一个整体的认识与了解，熟练掌握 MySQL 数据库的安装、配置与管理。
- 第 2、3 章主要讲解数据库的基本操作，包括数据库、数据表，数据的增、删、改、查，创建数据表时数据类型及表约束的选取，以及字符集与校对集在 MySQL 中的作用。此部分是所有想要使用 MySQL 的初学者都必须掌握的内容。
- 第 4 章从数据库实际运用的角度，讲解项目开发中如何设计一个合理、规范和高效的数据库。主要包括数据库设计的 6 个阶段、涉及的相关人员、数据库设计的三范式以及数据库建模工具的使用，并以电子商务网站的数据库设计为例，演示如何根

据实际需求，设计出合理的数据库。

- 第 5、6 章主要从数据操作的角度，讲解如何快速复制表结构、添加表数据，如何对数据进行判断、分组、排序与限量，如何连接多个数据表查询数据，如何建立外键约束等。此部分是所有想要从事与数据库开发相关工作的人员必须掌握的操作内容。
- 第 7 章从数据库安全的角度，讲解为操作数据库的用户分配权限的重要性。提供创建用户、分配密码、授予以及回收权限等具体的 SQL 操作。通过本章的学习，要求读者能够熟练操作数据库的同时保证数据的安全。
- 第 8～11 章从多角度讲解数据库优化的方式，包括存储引擎的选取，视图、事务、索引、锁和预处理 SQL 语句的应用，函数、变量、存储过程、游标、触发器和事件等多语句的编程，分表分区技术、数据碎片整理、慢查询日志、查询缓存等常见的优化解决方案。此部分内容有助于读者循序渐进地掌握如何提升和改进 MySQL 的性能。
- 第 12 章介绍在 Linux 系统中 MySQL 的安装与配置，MySQL 的多实例部署、数据备份与还原，以及如何利用主从复制或组复制来提高数据库的可用性和负载能力。

在上面所列举的 12 章中，第 1～6 章讲解 MySQL 的基础理论与 SQL 指令，主要帮助初学者奠定扎实的基本功；第 7～12 章从安全与优化角度深层次挖掘 MySQL，提升读者的 MySQL 运用技能，积累开发经验。

在学习过程中，读者一定要亲自动手实践本书中的案例，如果不能完全理解书中所讲知识，读者可以登录“博学谷”平台，通过平台中的教学视频进行深入学习。学习完一个知识点后，要及时在“博学谷”平台进行测试，以巩固学习内容。

另外，如果读者在理解知识点的过程中遇到困难，建议不要纠结于某个地方，可以先往后学习。通常来讲，通过逐渐地学习，前面不懂和疑惑的知识也就能够理解了。在学习的过程中，一定要多动手实践，如果在实践的过程中遇到问题，建议多思考，厘清思路，认真分析问题发生的原因，并在问题解决后总结经验。

致谢

本书的编写和整理工作由传智播客教育科技股份有限公司完成，主要参与人员有吕春林、韩冬、乔治铭、陈欢、李东超、韩振国、王金涛、孙书华等，全体人员在这近一年的编写过程中付出了很多辛勤的汗水，在此一并表示衷心的感谢。

意见反馈

尽管我们付出了最大的努力，但教材中难免会有欠妥之处，欢迎各界专家和读者朋友提出宝贵意见，我们将不胜感激。您在阅读本书时，如发现任何问题或有不认同之处可以通过电子邮件与我们取得联系。

请发送电子邮箱至 itcast_book@vip.sina.com。

黑马程序员

2019 年 2 月于北京

目 录

第1章 数据库入门

学习目标

- 掌握数据库、数据库系统、数据库管理系统的基本概念
- 了解数据库技术发展经历的三个阶段
- 掌握关系模型、SQL 的基本概念
- 掌握 MySQL 的安装、配置、启动、登录等操作
- 了解常用图形化工具的使用

数据库技术是计算机应用领域中非常重要的技术。它产生于 20 世纪 60 年代末，是数据管理的最新技术，也是软件技术的一个重要分支。本章重点讲解数据库的基础知识以及 MySQL 的安装与使用。

1.1 数据库基础知识

1.1.1 数据库概述

数据库(Database，DB)是按照数据结构来组织、存储和管理数据的仓库，其本身可被看作电子化的文件柜，用户可以对文件中的数据进行增加、删除、修改、查找等操作。需要注意的是，这里所说的数据(Data)不仅包括普通意义上的数字，还包括文字、图像、声音等。也就是说，凡是在计算机中用来描述事物的信息都可称为数据。

数据库技术是计算机领域重要的技术之一。在互联网、银行、通信、政府部门、企事业单位、科研机构等领域，都存在着大量的数据。数据库技术研究如何对数据进行有效的管理，包括组织和存储数据，在数据库系统中减少数据存储冗余、实现数据共享、保障数据安全，以及高效地检索和处理数据。

大多数初学者认为数据库就是数据库系统(Database System，DBS)。其实，数据库系统的范围比数据库大很多。数据库系统是指在计算机系统中引入数据库后的系统，除了数据库，还包括数据库管理系统(Database Management System，DBMS)、数据库应用程序等。为了让读者更好地理解数据库系统，下面通过一张图来描述，如图 1-1 所示。

图 1-1 描述了数据库系统的几个重要部分，如数据库、数据库管理系统、数据库应用程序等，具体解释如下。

(1) 数据库。数据库提供了一个存储空间用来存储各种数据，可以将数据库视为一个存储数据的容器。

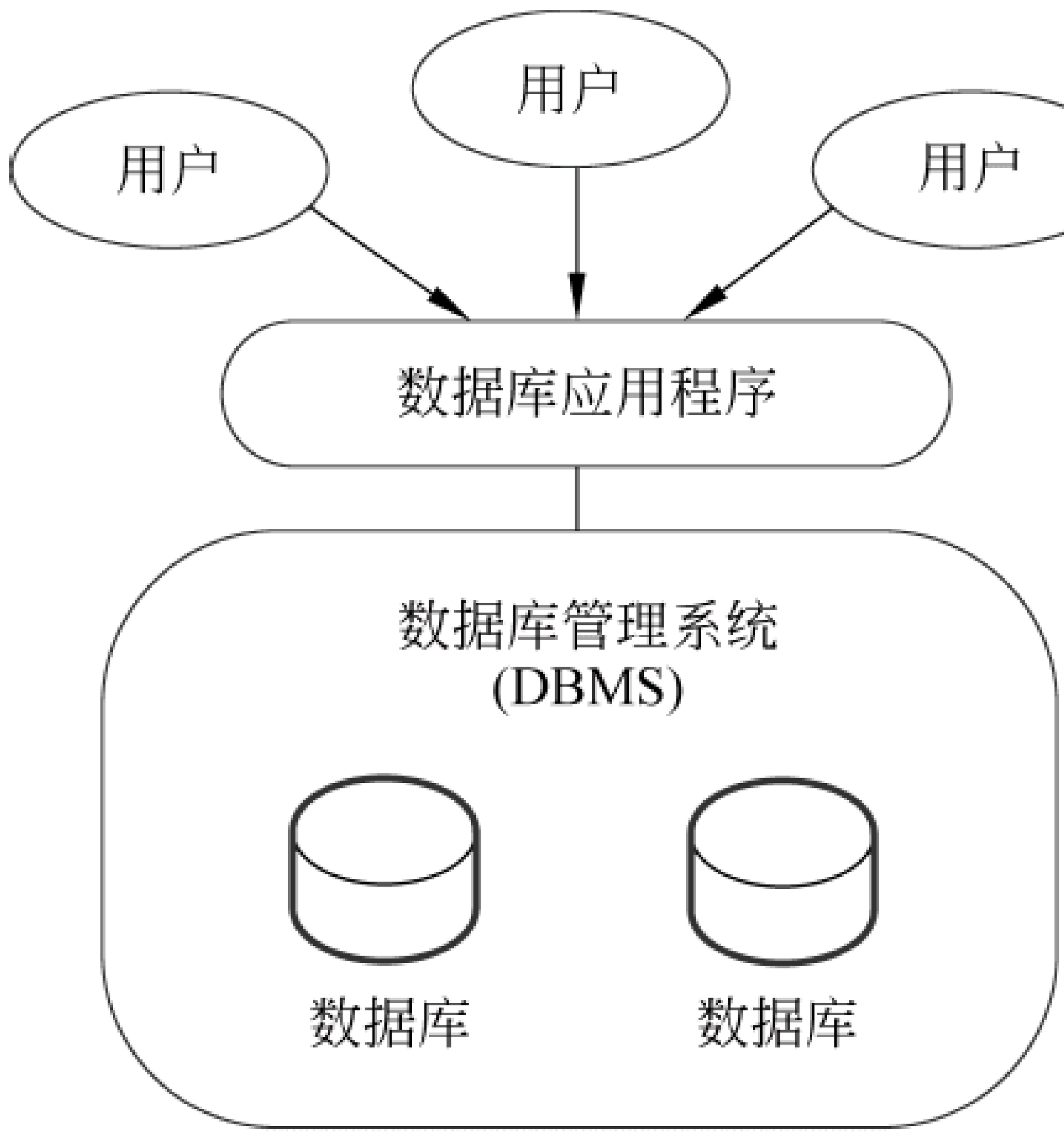


图 1-1 数据库系统

(2) 数据库管理系统。专门用于创建和管理数据库的一套软件，介于应用程序和操作系统之间，如 MySQL、Oracle、SQL Server、DB2 等。数据库管理系统不仅具有最基本的数据管理功能，还能保证数据的完整性、安全性和可靠性。

(3) 数据库应用程序。虽然已经有了数据库管理系统，但在很多情况下，数据库管理系统无法满足用户对数据库的管理。此时，就需要使用数据库应用程序与数据库管理系统进行通信、访问和管理 DBMS 中存储的数据。

1.1.2 数据库技术的发展

任何一种技术都不是凭空产生的，而是经历了长期的发展过程。通过了解数据库技术的发展历史，可以理解现在的数据库技术是基于什么样的需求而诞生的。

数据库技术的发展主要分为 3 个阶段，分别是人工管理阶段、文件系统阶段和数据库系统阶段。关于这 3 个阶段的具体介绍如下。

1. 人工管理阶段

在 20 世纪 50 年代中期以前，计算机主要用于科学计算，硬件方面没有磁盘等直接存取设备，只有磁带、卡片和纸带；软件方面没有操作系统和管理数据的软件。数据的输入、存取等，需要人工操作。人工管理阶段处理数据非常麻烦和低效，该阶段具有如下特点。

(1) 数据不在计算机中长期保存。

(2) 没有专门的数据管理软件，数据需要应用程序自己管理。

(3) 数据是面向应用程序的，不同应用程序之间无法共享数据。

(4) 数据不具有独立性，完全依赖于应用程序。

2. 文件系统阶段

从 20 世纪 50 年代后期到 60 年代中期，硬件方面有了磁盘等直接存取设备，软件方面有了操作系统，数据管理进入了文件系统阶段。这个阶段，数据以文件为单位保存在外存储器上，由操作系统管理，程序和数据分离，实现了以文件为单位的数据共享。

文件系统阶段具有如下特点。

（1）数据在计算机的外存设备上长期保存，可以对数据反复进行操作。

（2）通过文件系统管理数据，文件系统提供了文件管理功能和存取方法。

（3）虽然在一定程度上实现了数据独立性和共享性，但都非常薄弱。

3. 数据库系统阶段

从 20 世纪 60 年代后期开始，计算机应用越来越广泛，管理的数据量越来越多，同时对多种应用程序之间数据共享的需求越来越强烈，文件系统的管理方式已经无法满足需求。为了提高数据管理的效率，解决多用户、多应用程序共享数据的需求，数据库技术应运而生，由此进入了数据库系统阶段。

数据库系统阶段具有如下特点。

（1）数据结构化。数据库系统实现了整体数据的结构化，这是数据库主要的特征之一。这里所说的“整体”结构化，是指在数据库中的数据不只是针对某个应用程序，而是面向整体的。

（2）数据共享。因为数据是面向整体的，所以数据可以被多个用户、多个应用程序共享使用，可以大幅度地减少数据冗余，节约存储空间，避免数据之间的不相容性与不一致性。

例如，企业为所有员工统一配置即时通讯和电子邮箱软件，若两种软件的用户数据（如员工姓名、所属部门、职位等）无法共享，就会出现如下问题。

① 两种软件各自保存自己的数据，数据结构不一致，无法互相读取。软件的使用者需要向两个软件分别录入数据。

② 由于相同的数据保存两份，会造成数据冗余，浪费存储空间。

③ 若修改其中一份数据，忘记修改另一份数据，就会造成数据的不一致。

使用数据库系统后，数据只需保存一份，其他软件都通过数据库系统存取数据，就实现了数据的共享，解决了前面提到的问题。

（3）数据独立性高。数据的独立性包含逻辑独立性和物理独立性。其中，逻辑独立性是指数据库中数据的逻辑结构和应用程序相互独立，物理独立性是指数据物理结构的变化不影响数据的逻辑结构。数据的独立性原理会在三级模式和二级映像中详细讲解。

（4）数据统一管理与控制。数据的统一控制包含安全控制、完整控制和并发控制。简单来说就是防止数据丢失、确保数据正确有效，并且在同一时间内，允许用户对数据进行多路存取，防止用户之间的异常交互。例如，春节期间网上订票时，由于出行人数多、时间集中和抢票的问题，火车票数据在短时间内会发生巨大的变化，数据库系统要保证数据不能出现问题。

1.1.3　三级模式和二级映像

美国国家标准学会（American National Standards Institute，ANSI）所属的标准计划与需求委员会（Standards Planning and Requirements Committee，SPARC）在 1971 年公布的研究报告中提出了 ANSI-SPARC 体系结构，即三级模式结构（或称为三层体系结构）。ANSI-SPARC 最终没有成为正式标准，但它仍然是理解数据库管理系统的基础。

三级模式是指数据库管理系统从三个层次来管理数据，分别是外部层（External

Level)、概念层(Conceptual Level)和内部层(Internal Level)。这三个层次分别对应三种不同类型的模式,分别是外模式(External Schema)、概念模式(Conceptual Schema)和内模式(Internal Schema)。在外模式与概念模式之间,以及概念模式与内模式之间,还存在映像,即二级映像,具体如图 1-2 所示。

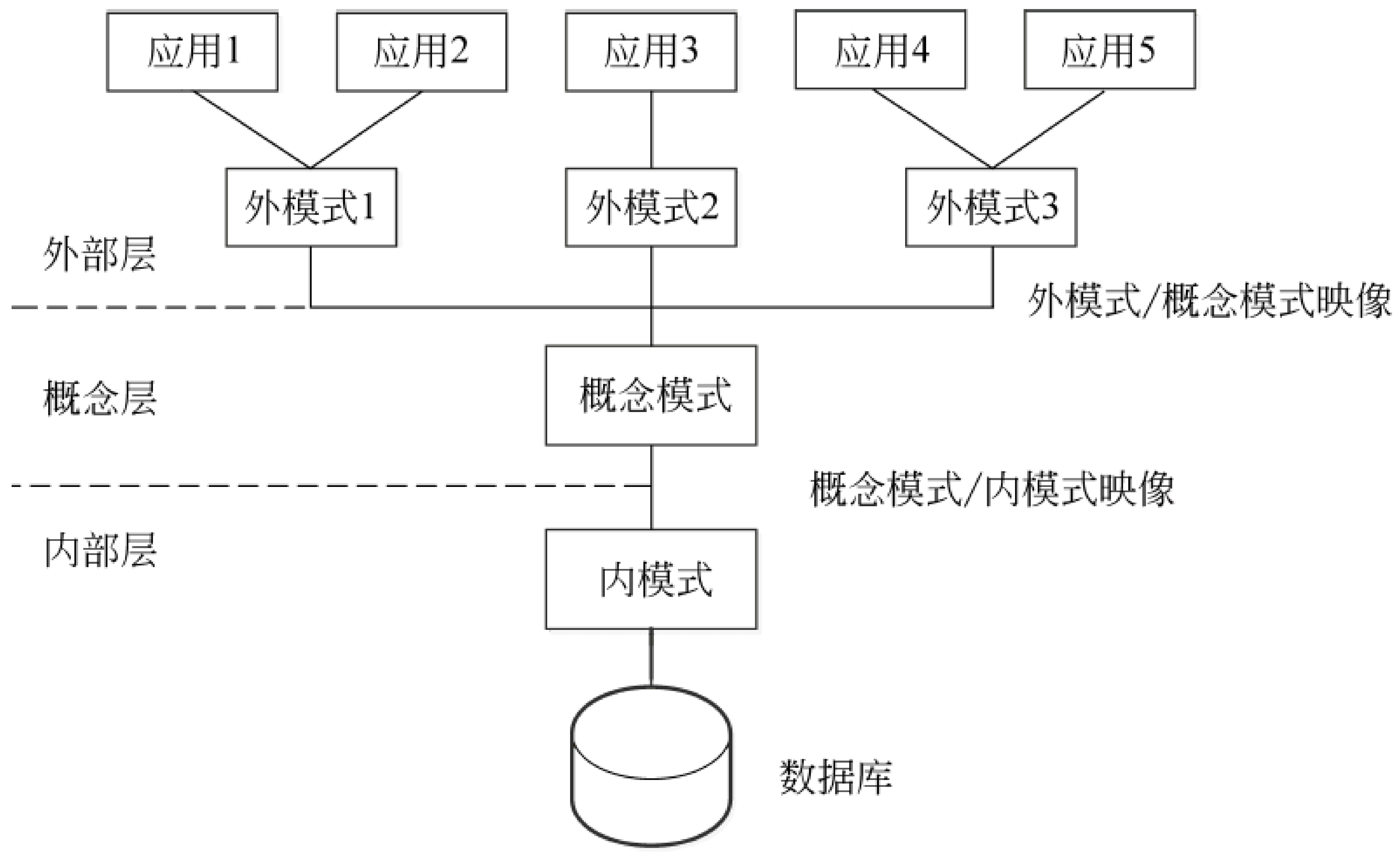


图 1-2　三级模式和二级映像

在图 1-2 中,外模式面向应用程序,描述用户的数据视图(View);内模式(又称为物理模式、存储模式)面向物理上的数据库,描述数据在磁盘中如何存储;概念模式(又称为模式、逻辑模式)面向数据库设计人员,描述数据的整体逻辑结构。

由于三级模式比较抽象,为了更好地理解,下面将计算机中常用的 Excel 电子表格类比成数据库,并假设有一个商城使用电子表格来保存商品信息。

(1) 概念模式。概念模式类似于表格的列标题,它描述了商品表中包含哪些信息,如图 1-3 所示。

编号	商品名称	商品分类	商品价格	库存	销量

图 1-3　商品信息表格

在图 1-3 中,表的横向称为行,纵向称为列,第一行就是列标题,用来描述该列的数据表示什么含义。实际上,概念模式在数据库中描述的信息还有很多,如多张表之间的联系、表中每一列的数据类型和长度等,读者在后面的学习中就会接触到这些内容。

(2) 内模式。在将 Excel 表格另存为文件时,可以选择保存的文件路径、保存类型(如 XLS、XLSX、CSV 等格式)等,这些与存储相关的描述信息相当于内模式。在数据库中,内模式描述数据的物理结构和存储方式,如堆文件、索引文件、散列(Hash)文件等。

(3) 外模式。在打开一个电子表格后,默认会显示表格中所有的数据,这个表格称为基本表。在将数据提供给其他用户时,出于权限、安全控制等因素的考虑,只允许用户看到一部分数据,或不同用户看到不同的数据,这样的需求就可以用视图来实现。图 1-4 演示了视

图和基本表的关系。

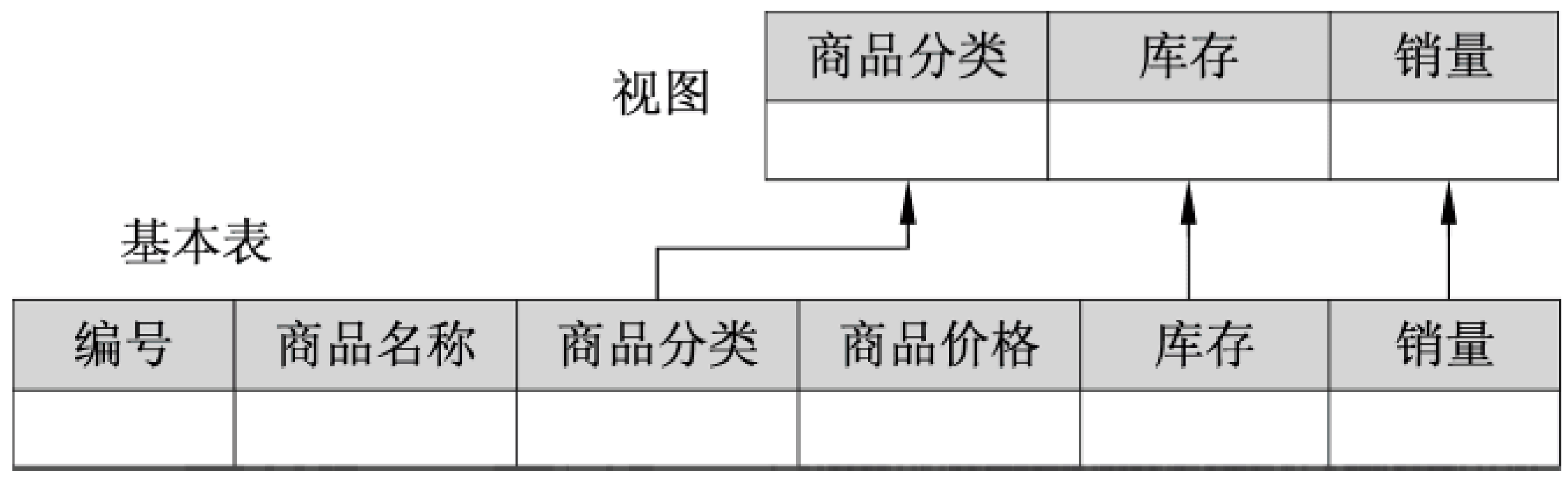


图 1-4　视图与基本表

在图 1-4 中，基本表中的数据是实际存储在数据库中的，而视图中的数据是查询或计算出来的。由此可见，外模式可以为不同用户的需求创建不同的视图，且由于不同用户的需求不同，数据的显示方式也会多种多样。因此，一个数据库中会有多个外模式，而概念模式和内模式则只有一个。

通过前面的分析可知，三级模式是数据的三个抽象级别，每个级别关心的重点不同。为了使三级模式之间产生关联，数据库管理系统在三级模式之间提供了二级映像功能。二级映像是一种规则，它规定了映像双方如何进行转换。通过二级映像，体现了逻辑和物理两个层面的数据独立性。具体解释如下。

（1）逻辑独立性。外模式/概念模式映像体现了逻辑独立性。逻辑独立性是指当修改了概念模式，不影响其上一层的外模式。例如，将图 1-4 中基本表的“库存”和“销量”拆分到另一张表中，此时概念模式发生了更改，但可以通过改变外模式/概念模式的映像，继续为用户提供原有的视图，如图 1-5 所示。

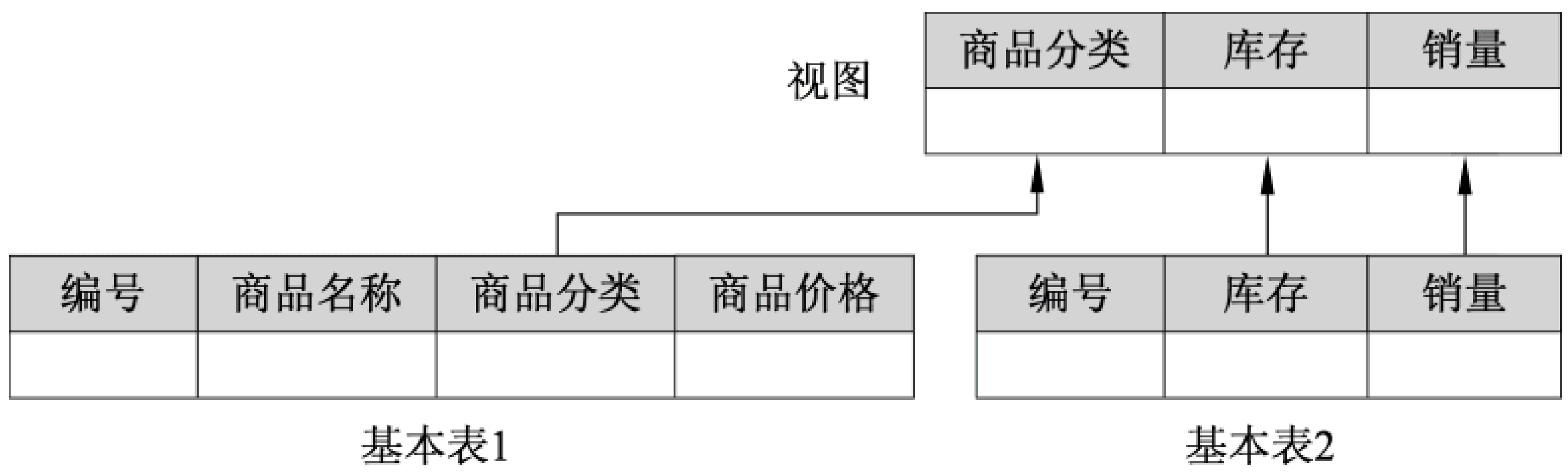


图 1-5　视图与基本表

由此可见，逻辑独立性能够让使用视图的用户感觉不到基本表的改变。其实，逻辑独立性带来的好处还有很多，随着后面的学习，读者会有更深入的体会。

（2）物理独立性。概念模式/内模式映像体现了物理独立性。物理独立性是指修改了内模式，不影响其上层的概念模式和外模式。例如，在 Excel 中将.xls 文件另存为.xlsx 文件，虽然更换了文件格式，但是打开文件后显示的表格内容一般不会发生改变。

在数据库中，更换更先进的存储结构，或者创建索引以加快查询速度，内模式会发生改变。此时，只需改变概念模式/内模式映像，就不会影响到原有的概念模式。

另外，物理独立性使得用户不必了解数据库内部的存储原理，即可正常使用数据库来保存数据。数据库管理系统会自动将用户的操作转换成物理级数据库的操作。

多学一招：数据库相关的人员

数据库系统涉及一些人员，主要包括数据库管理员(Database Administrator，DBA)、应

用程序员（Application Programmer）和最终用户（End User）。关于这些人员的具体解释如下。

（1）数据库管理员。负责管理和维护数据库，参与数据库的设计、测试和部署。数据库管理员需要对数据库系统非常精通，且应具有较高的技术水平和较深的资历。

（2）应用程序员。负责为最终用户设计和编写程序，并进行调试和安装，以便最终用户利用应用程序来对数据库进行存取操作。

（3）最终用户。一般为非计算机专业人员，通过应用程序访问数据库。例如，在12306网站订票的用户、在一些购物网站购买商品的用户，他们可能对数据库完全不了解，在使用浏览器、客户端等应用程序时，间接地访问了数据库。

1.1.4 数据模型

数据库的类型通常按照数据模型（Data Model）来划分。数据模型是数据库系统的核心和基础，它是对现实世界数据特征的抽象，用来描述数据，可以理解成一种数据结构。

在数据库的发展过程中，出现了3种基本数据模型，分别是层次模型（Hierarchical Model）、网状模型（Network Model）和关系模型（Relational Model）。目前使用最多的数据模型是关系模型，建立在关系模型基础上的数据库称为关系数据库。本书讲解的 MySQL 就是一个关系数据库管理系统。

若要理解关系模型，首先应该掌握一些理论知识，下面进行详细讲解。

1. 数据建模

数据建模是对现实世界中的各类数据的抽象组织，以确定数据库的管辖范围、数据的组织形式等。数据建模大致分为3个阶段，分别是概念建模阶段、逻辑建模阶段、物理建模阶段，相应的产物分别是概念模型、逻辑模型和物理模型，具体如图1-6所示。

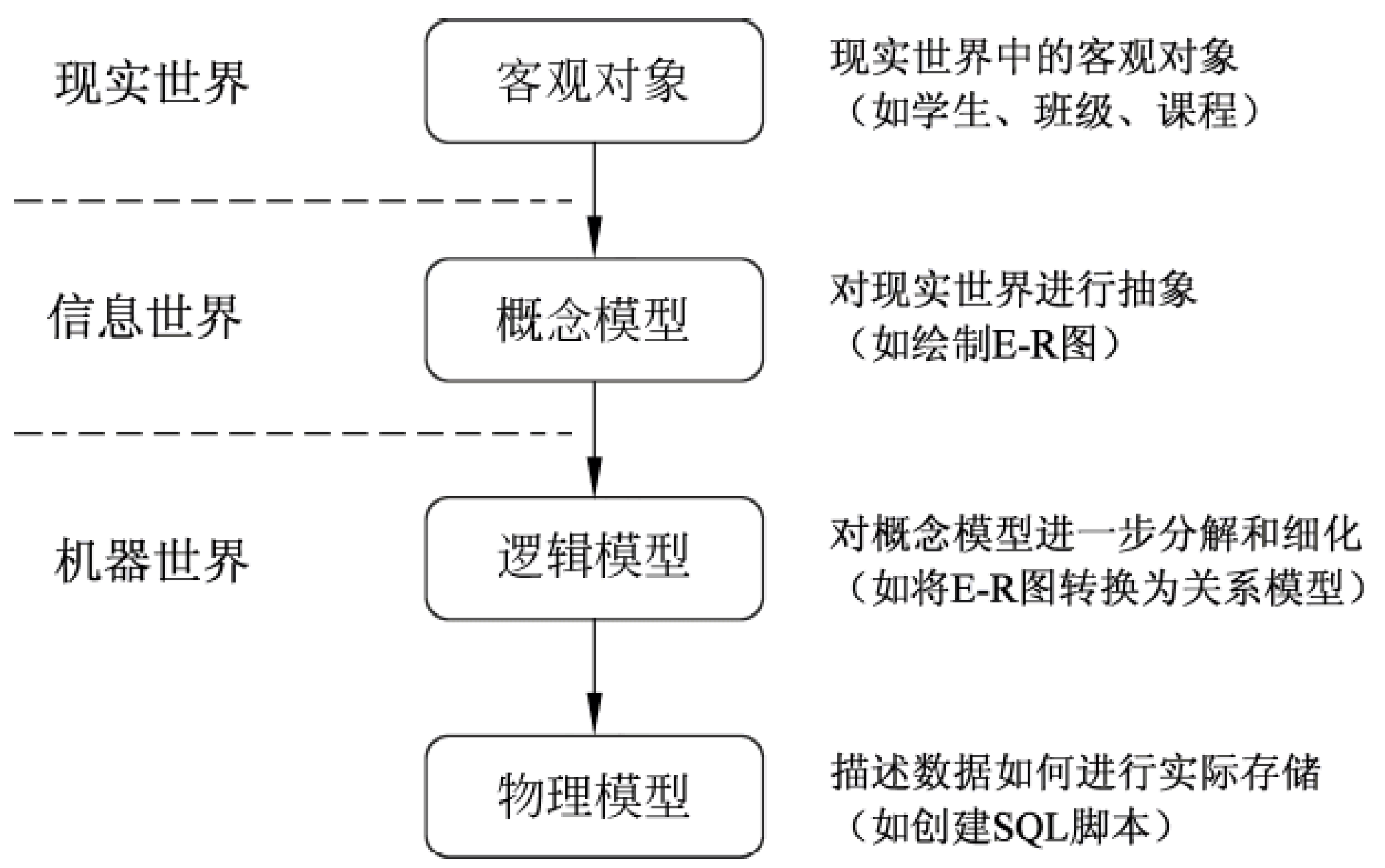


图 1-6　数据建模

在图1-6中，概念模型是现实世界到机器世界的中间层，它将现实世界中的客观对象（如学生、班级、课程）抽象成信息世界的数据。逻辑模型是指数据的逻辑结构，可以选择层次模型、网状模型或关系模型。在完成逻辑模型后，最后使用物理模型描述数据如何进行实际存储，也就是将逻辑模型转换成计算机能够识别的模型。

2. 概念模型的术语

在概念模型中可以通过一些术语来对现实世界进行抽象,具体如下。

(1) 实体(Entity)。实体是指客观存在并可相互区分的事物,如学生、班级、课程。

(2) 属性(Attribute)。属性是指实体所具有的某一特性,一个实体可以由若干个属性来描述。例如,学生实体的属性有学号、学生姓名和学生性别。属性由两部分组成,分别是属性名和属性值。例如,学号、学生姓名和学生性别是属性名,而"1、张三、男"这些具体值是属性值。

(3) 联系(Relationship)。这里所说的联系是指实体与实体之间的联系,有一对一、一对多、对多对三种情况。例如,每个学生都有一个学生证,学生和学生证之间是一对一的联系;一个班级有多个学生,班级和学生是一对多的联系;一个学生可以选修多门课程,一门课程又可以被多个学生选修,学生和课程之间就形成了多对多的联系。

(4) 实体型(Entity Type)。实体型即实体类型,通过实体名(如学生)及其属性名集合(如"学号、学生姓名、学生性别")来抽象描述同类实体。

(5) 实体集(Entity Set)。实体集是指同一类型的实体集合,如全校学生就是一个实体集。

3. E-R 图

E-R 图也称为实体-联系图(Entity Relationship Diagram),是一种用图形表示的实体联系模型,由 Peter Chen 于 1976 年提出。E-R 图提供了表示实体型、属性和联系的方法,用来描述现实世界的概念模型。其通用的表示方式如下。

- 实体:用矩形框表示,将实体名写在框内。
- 属性:用椭圆框表示,将属性名写在框内,用连线将实体与属性连接。
- 联系:用菱形框表示,将联系名写在框内,用连线将相关的实体连接,并在连线旁标注联系类型(一 0 对一"1 : 1"、一对多"1 : n"、多对多"n : m")。

下面演示学生与班级、学生与课程的 E-R 图,分别如图 1-7 和图 1-8 所示。

从图 1-7 和图 1-8 中可以看出,E-R 图接近于普通人的思维,即使不具备计算机专业知识,也可以理解其表示的含义。

4. 关系模型

关系模型由 IBM 公司研究员 Edgar Frank Codd 于 1970 年发表的论文中提出,经过多年的发展,已经成为目前最常用、最重要的模型之一。

在关系模型中有一些基本的概念,具体如下。

(1) 关系(Relation)。关系一词与数学领域有关,它是集合基础上的一个重要的概念,用于反映元素之间的联系和性质。从用户角度来看,关系模型的数据结构是二维表,即通过二维表来组织数据。一个关系对应一张二维表,表中的数据包括实体本身的数据和实体间的联系。

下面通过图 1-9 演示一个简单的学生信息二维表。

(2) 属性(Attribute)。二维表中的列称为属性,每个属性都有一个属性名。

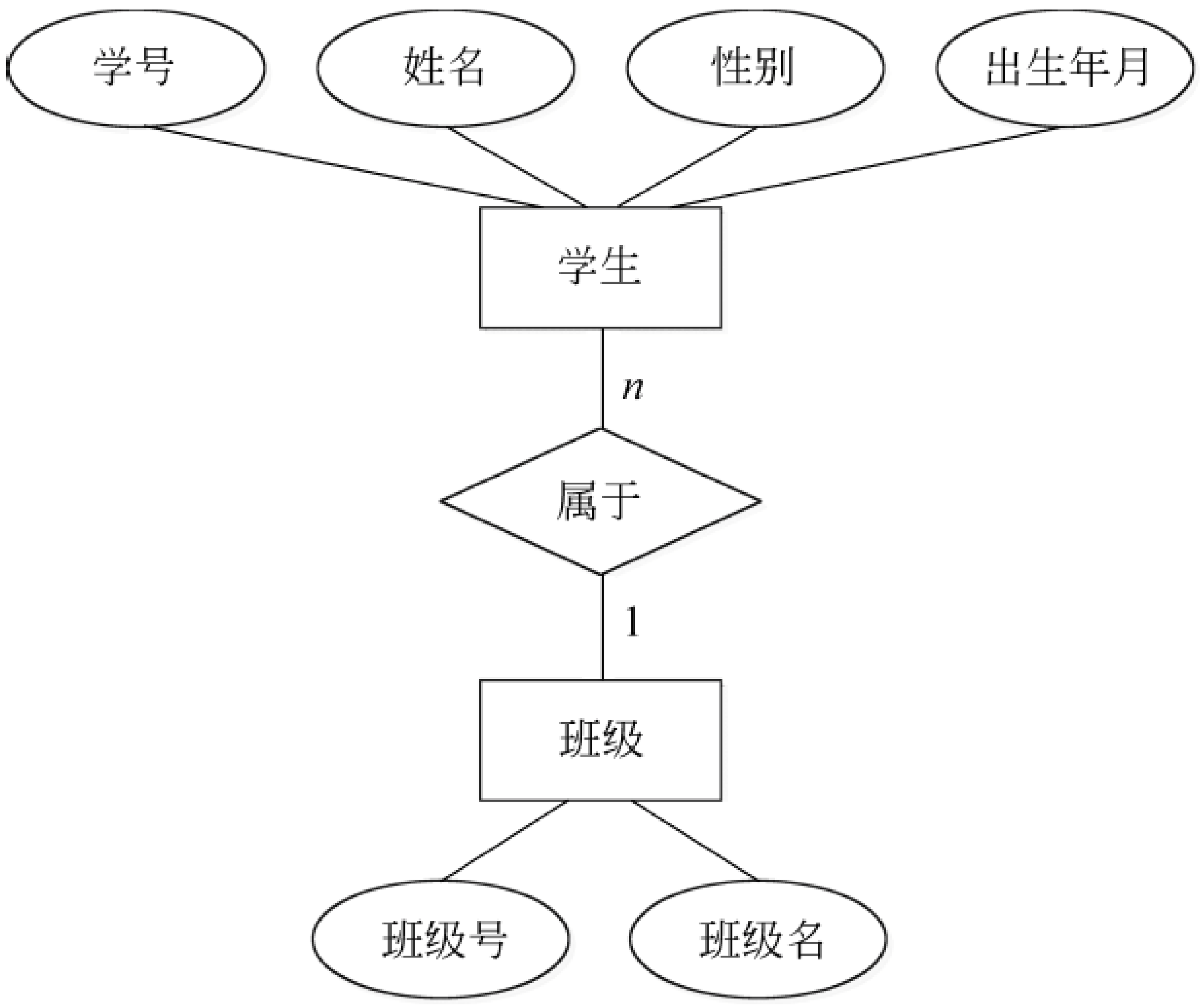


图 1-7 学生与班级 E-R 图

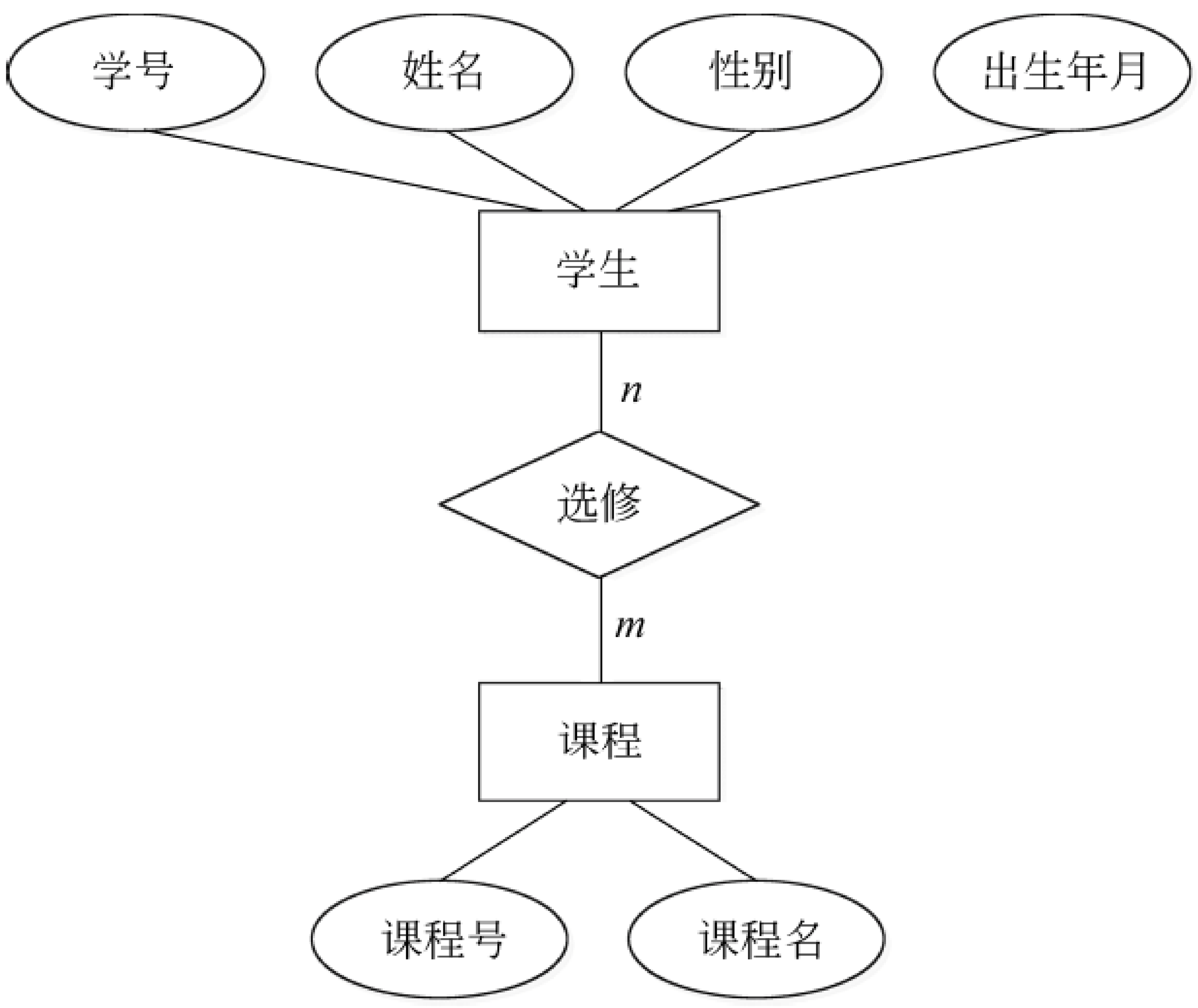


图 1-8 学生与课程 E-R 图

属性（字段）

学号	姓名	性别	出生年月
1	张三	男	1996-02
2	李四	女	1996-04
3	小明	男	1996-06

元组（记录）

图 1-9 学生信息二维表

(3) 元组(Tuple)。二维表中的每一行数据称为一个元组。

小提示：根据不同的习惯，属性也可以称为字段(Field)，元组也可以称为记录(Record)。

(4) 域(Domain)。域是指属性的取值范围，例如，性别属性的域为男、女。

(5) 关系模式(Relation Schema)。关系模式是关系的描述,通常可以简记为“关系名(属性 1,属性 2,…,属性 n)”。例如,图 1-9 中的二维表的关系模式如下。

```
学生 (学号,姓名,性别,出生年月)
```

(6) 键(Key)。在二维表中,若要唯一标识某一条记录,需要用到键(又称为关键字、码)。例如,学生的学号具有唯一性,学号可以作为学生实体的键。而学生姓名可能存在重名,不适合作为键。通过键可以为两张表建立联系,如图 1-10 所示。

学生表

学号	学生姓名	学生性别	班级号
1	张三	男	1
2	李四	女	1
3	小明	男	2
4	小红	女	2

班级表

班级号	班级名称	班主任
1	软件班	张老师
2	设计班	王老师

图 1-10　学生表与班级表

在图 1-10 中,班级表中的“班级号”是该表的键,学生表中的“班级号”表示学生所属的班级,两者建立了一对多的联系,即一个班级中有多个学生。其中,班级表的“班级号”称为主键(Primary Key),学生表的“班级号”称为外键(Foreign Key)。

学生与课程的多对多联系,可以通过中间表来实现,如图 1-11 所示。

课程表

课程号	课程名
1	计算机
2	数据库

学生选课表

课程号	学号
1	2
1	3
2	2
2	3

学生表

学号	学生姓名	学生性别	班级号
1	张三	男	1
2	李四	女	1
3	小明	男	2
4	小红	女	2

图 1-11　学生表与课程表

在图 1-11 中,学生表与课程表之间通过学生选课表关联。学生选课表将学生与课程的多对多关系拆解成两个一对多关系,即一个学生选修多门课,一门课被多个学生选修。

5. 关系模型的完整性

为了保证数据库中数据的正确性和相容性,需要对关系模型进行完整性约束。完整性通常包括实体完整性、参照完整性和用户自定义完整性,具体解释如下。

(1) 实体完整性。实体完整性要求关系中的主键不能重复,且不能取空值。空值是指不知道、不存在或无意义的值。由于关系中的元组对应现实世界中互相之间可区分的个体,这些个体使用主键来唯一标识,若主键为空或重复,则无法唯一标识每个个体。

(2) 参照完整性。参照完整性要求关系中的外键要么取空值,要么取被参照关系中的某个元组的主键值。例如,通过学生所属的班级号可以找到对应的班级,这就符合参照完整性;而如果对应的班级被删除了,学生通过班级号找不到班级,就不符合参照完整性。

(3) 用户自定义完整性。用户自定义完整性是用户针对具体的应用环境定义的完整性约束条件，由 DBMS 检查用户自定义的完整性。例如，用户名不允许重复。

1.1.5　关系运算

关系模型可以使用关系代数(Relational Algebra)来进行关系运算。关系代数是一种抽象的查询语言，是研究关系模型的数学工具。关系代数运算符主要包括并、差、交、笛卡儿积、选择、投影、连接和除，具体如表 1-1 所示。

表 1-1　关系代数运算符

集合运算符	含　义	关系运算符	含　义
∪	并	σ	选择
－	差	π	投影
∩	交	⋈	连接
×	笛卡儿积	÷	除

接下来针对表 1-1 列举的这些关系代数运算符进行详细讲解。

1. 并(Union)、差(Difference)、交(Intersection)

并、差、交运算要求参与运算的两个关系具有相同数量的属性，其运算结果是一个具有相同数量属性的新关系。设有关系 R 和关系 S，R∪S 表示合并两个关系中的元组，R－S 表示找出属于 R 但不属于 S 的元组，R∩S 表示找出既属于 R 又属于 S 的元组。

下面通过图 1-12 演示 R∪S、R－S 和 R∩S 的运算结果。

R

学号	学生姓名
1	张三
2	李四

S

学号	学生姓名
1	张三
3	小明

R∪S

学号	学生姓名
1	张三
2	李四
3	小明

R−S

学号	学生姓名
2	李四

R∩S

学号	学生姓名
1	张三

图 1-12　并、差、交

2. 笛卡儿积(Cartesian Product)

设关系 R 有 n 个属性，关系 S 有 m 个属性，R 和 S 的笛卡儿积结果是一个具有 $n+m$ 个属性的新关系。在新关系中，元组的前 n 个属性来自 R，后 m 个属性来自 S，元组的总个数是 R 和 S 中的元组的乘积。下面通过图 1-13 演示 R×S 的运算结果。

3. 选择(Selection)、投影(Projection)

选择是在一个关系中将满足条件的元组找出来，即水平方向筛选；投影是在一个关系中去掉不需要的属性，保留需要的属性，即垂直方向筛选。

R

学号	学生姓名
1	张三
2	李四

S

班级号	班级名称
1	软件班
2	设计班

R×S

学号	学生姓名	班级号	班级名称
1	张三	1	软件班
1	张三	2	设计班
2	李四	1	软件班
2	李四	2	设计班

图 1-13　笛卡儿积

下面通过图 1-14 演示选择和投影操作。其中，选择操作 $\sigma_{学号=1}$(R)表示在关系 R 中查找学号为 1 的学生；投影操作 $\pi_{学号,学生姓名}$(R)表示在关系 R 中查找学号和学生姓名。

R

学号	学生姓名	学生性别
1	张三	男
2	李四	女

$\sigma_{学号=1}$(R)

学号	学生姓名	学生性别
1	张三	男

$\pi_{学号,学生姓名}$(R)

学号	学生姓名
1	张三
2	李四

图 1-14　选择和投影

4. 连接(Join)

连接是在两个关系的笛卡儿积中选取属性间满足一定条件的元组。由于笛卡儿积的结果可能会包含很多没有意义的元组，所以相比之下连接运算更为实用。

常用的连接方式有等值连接(Equi-Join)和自然连接(Natural Join)。设有关系 R 和关系 S，使用 A 和 B 分别表示 R 和 S 中数目相等且可比的属性组。等值连接是在 R 和 S 的笛卡儿积中选取 A、B 属性值相等的元组。自然连接是一种特殊的等值连接，要求 R 和 S 必须有相同的属性组，进行等值连接后再去除重复的属性组。

下面通过图 1-15 演示等值连接和自然连接的结果。

R

学号	学生姓名	班级号
1	张三	1
2	李四	1
3	小明	2
4	小红	2

S

班级号	班级名称
1	软件班
2	设计班
3	网络班

$R \underset{R.班级号=S.班级号}{\bowtie} S$ (等值连接)

学号	学生姓名	R.班级号	S.班级号	班级名称
1	张三	1	1	软件班
2	李四	1	1	软件班
3	小明	2	2	设计班
4	小红	2	2	设计班

$R \bowtie S$ (自然连接)

学号	学生姓名	班级号	班级名称
1	张三	1	软件班
2	李四	1	软件班
3	小明	2	设计班
4	小红	2	设计班

图 1-15　等值连接和自然连接

5. 除(Division)

如果把笛卡儿积看作乘法运算,则除法是笛卡儿积的逆运算。设有关系R和关系S,除运算需满足S的属性集是R属性集的真子集,R÷S的结果是R属性集减去S属性集的结果。例如,R(A,B,C,D)÷S(C,D)的结果由A和B两个属性构成。

下面通过图1-16演示除运算的结果。

R

课程号	学号
1	2
2	2
3	2
1	3
2	3
1	4

S1

学号
2

S2

学号
2
3

R÷S1

课程号
1
2
3

R÷S2

课程号
1
2

图1-16　除运算

在图1-16中,R是学生选课表,R÷S1表示查询学号为2的学生所选的课程,R÷S2表示查询学号为2和3的学生共同选择的课程。

为了更好地理解除运算的过程,可以转换成其他关系代数来实现。以图1-16中的R÷S2为例,其运算过程如图1-17所示。

R

课程号	学号
1	2
2	2
3	2
1	3
2	3
1	4

S

学号
2
3

$\pi_{R-S}(R)$

课程号
1
2
3

$\pi_{R-S}(R)\times S$

课程号	学号
1	2
1	3
2	2
2	3
3	2
3	3

$A=(\pi_{R-S}(R)\times S)-R$

课程号	学号
3	3

$B=\pi_{A-S}(A)$

课程号
3

$\pi_{R-S}(R)-B$

课程号
1
2

图1-17　除运算的过程

在图1-17的运算中,一共经历了5个步骤,具体如下。

(1) 通过$\pi_{R-S}(R)$取出R和S中只有R拥有的课程号属性,并对元组去重。

(2) 通过$\pi_{R-S}(R)\times S$计算笛卡儿积。

(3) 通过$(\pi_{R-S}(R)\times S)-R$计算差集,设结果为A。

(4) 通过$\pi_{A-S}(A)$取出A中的课程号属性,设结果为B。

(5) 通过$\pi_{R-S}(R)-B$计算差集,得到的结果就是R÷S。

1.1.6　SQL语言

SQL(Structured Query Language,结构化查询语言)是一种数据库查询语言和程序设计语言,主要用于管理数据库中的数据,如存取数据、查询数据、更新数据等。

SQL 是 IBM 公司于 1975—1979 年开发出来的，在 20 世纪 80 年代，SQL 被美国国家标准学会（ANSI）和国际标准化组织（International Organization for Standardization，ISO）定义为关系数据库语言的标准。目前，各大数据库厂商的数据库产品从很大程度上支持了 SQL-92 标准，并在实践过程中对 SQL 标准作了一些修改和补充，所以不同数据库产品的 SQL 仍然存在少量的差别。

SQL 是由 4 部分组成的，具体如下。

（1）数据定义语言（Data Definition Language，DDL）。数据库定义语言主要用于定义数据库、表等。例如，CREATE 语句用于创建数据库、数据表等，ALTER 语句用于修改表的定义等，DROP 语句用于删除数据库、删除表等。

（2）数据操作语言（Data Manipulation Language，DML）。数据操作语言主要用于对数据库进行添加、修改和删除操作。例如，INSERT 语句用于插入数据，UPDATE 语句用于修改数据，DELETE 语句用于删除数据。

（3）数据查询语言（Data Query Language，DQL）。数据查询语言主要用于查询数据。例如，使用 SELECT 语句可以查询数据库中的一条数据或多条数据。

（4）数据控制语言（Data Control Language，DCL）。数据控制语言主要用于控制用户的访问权限。例如，GRANT 语句用于给用户增加权限，REVOKE 语句用于收回用户的权限，COMMIT 语句用于提交事务，ROLLBACK 语句用于回滚事务。

以上列举的 4 部分语言，在本书后面的几个章节中，会对其语法和使用进行详细讲解。读者此时只需了解 SQL 的基本组成部分即可。

小提示：SQL 支持数据库管理系统的三级模式结构，如图 1-18 所示。其中，外模式对应视图或部分基本表，概念模式对应基本表，内模式对应存储文件。

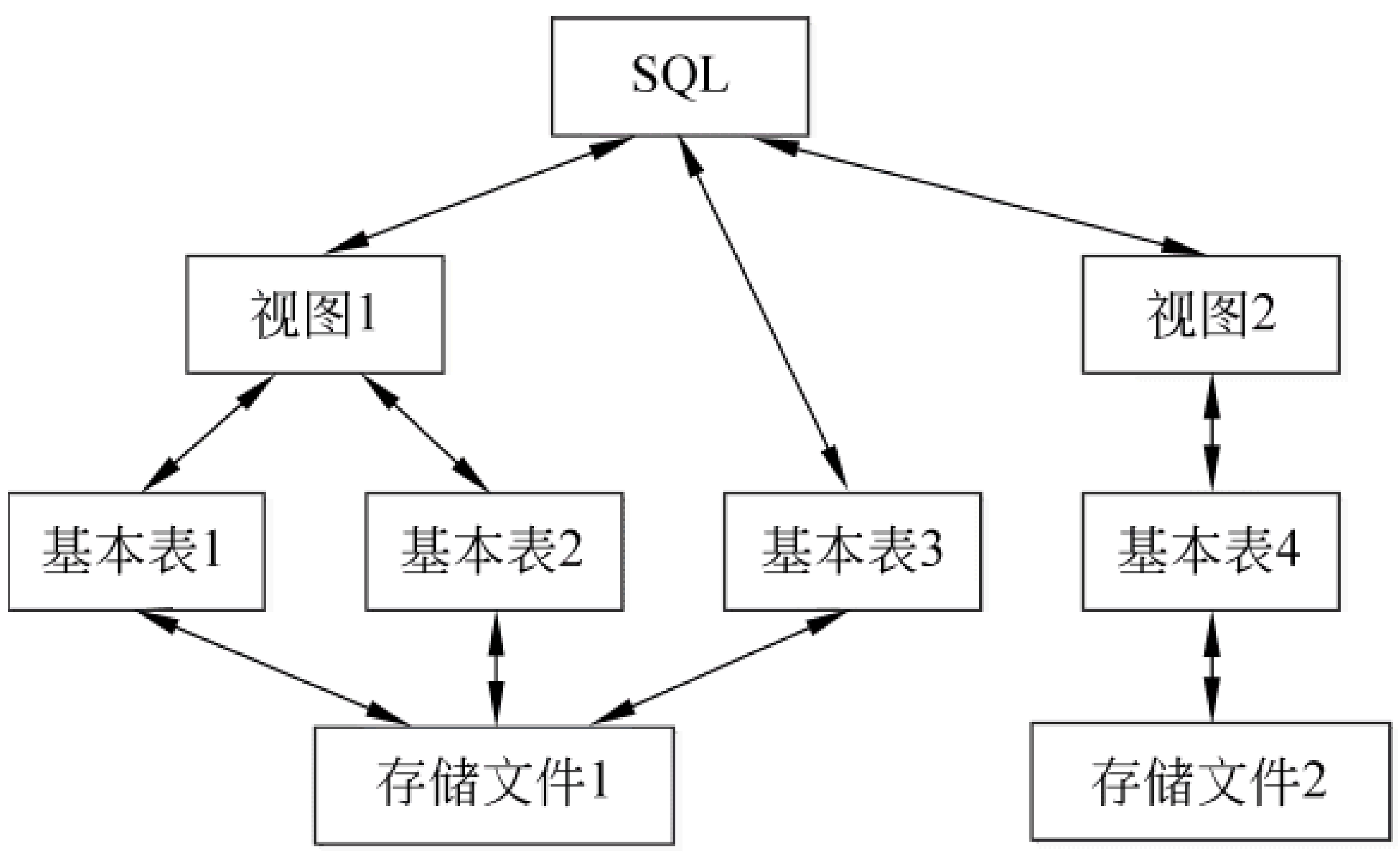


图 1-18　SQL 与三级模式

多学一招：数据库访问接口

数据库管理系统为访问数据库的应用程序提供了一些接口，用于执行 SQL 语句，对数据进行操作。在一些编程语言中，可以直接通过接口或类库进行使用。常见的有 Microsoft（微软）公司提出的 ODBC（Open DataBase Connectivity，开放数据库互连）、Java 语言的 JDBC（Java Database Connectivity，Java 数据库连接）、Microsoft. NET 框架中的 ADO .NET、PHP 语言中的 PDO（PHP Data Object）等。

1.1.7 常见的数据库产品

随着数据库技术的不断发展，关系数据库产品越来越多，常见的有 Oracle、SQL Server、MySQL 等，它们各自的特点如下所述。

1. Oracle

Oracle 数据库管理系统是由 Oracle(甲骨文)公司开发的，在数据库领域一直处于领先地位，市场占有率高，适用于各类大、中、小、微机环境，具有良好的兼容性、可移植性、可伸缩性，且性能高、安全性强。与 MySQL 相比，Oracle 虽然功能更加强大，但是软件的价格也比较高。

2. SQL Server

SQL Server 是 Microsoft 公司推出的关系数据库管理系统，它已广泛应用于电子商务、银行、保险、电力等行业，因易操作、界面良好等特点深受广大用户喜爱。早期版本的 SQL Server 只能在 Windows 平台上运行，而新版本的 SQL Server 2017 已经支持 Windows 和 Linux 平台。

3. DB2

DB2 是由 IBM 公司研制的关系数据库管理系统，主要应用于 UNIX(包括 IBM 的 AIX)、z/OS(适用于大型机的操作系统)、Windows Server 等平台下，具有较好的可伸缩性，可支持从大型计算机到单用户环境。DB2 提供了高层次的数据利用性、完整性、安全性和可恢复性，以及从小规模到大规模应用程序的执行能力，适合于海量数据的存储，但相对于其他数据库管理系统而言，DB2 的操作比较复杂。

4. MySQL

MySQL 是瑞典 MySQL AB 公司(先后被 Sun 和 Oracle 公司收购)开发的关系数据库管理系统，支持在 UNIX、Linux、Mac OS 和 Windows 等平台上使用。相对其他数据库而言，MySQL 体积小、速度快，使用更加方便、快捷，并且开放源代码，开发人员可根据需求自由进行修改。MySQL 采用社区版和商业版的双授权政策，兼顾了免费使用和付费服务的场景，软件使用成本低。因此，越来越多的公司开始使用 MySQL。尤其是在 Web 开发领域，MySQL 占据着举足轻重的地位。

多学一招：非关系数据库

互联网的高速发展对数据库技术提出了更高的要求，尤其对于超大规模和高并发类型的网站，传统关系数据库暴露了很多难以克服的问题。非关系数据库(Not Only SQL，NoSQL)的出现则弥补了关系数据库的不足，它的特点在于数据模型比较简单，灵活性强，性能非常高。常见的非关系数据库有 Redis、MongoDB，具体介绍如下。

(1) Redis。Redis 是一个高性能的非关系数据库产品，采用 key-value 的方式存储数据，适合用于内容缓存和处理大量数据的高负载访问，查询速度非常快。Redis 支持的数据类型包括 string(字符串)、hash(字典)、list(双向链表)、set(集合)和 zset(有序集合)，支持

持久化操作、主从同步等。

(2) MongoDB。MongoDB 是一个介于关系数据库和非关系数据库之间的产品，它在非关系数据库当中功能丰富，更接近关系数据库。它支持的数据结构非常松散，是类似 JSON 的 BSON 格式，可以存储比较复杂的数据类型。

MongoDB 最大的特点是它支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。不仅如此，它还是一个开源数据库，并且具有高性能、易部署、易使用、存储数据非常方便等特点。对于大数据量、高并发、弱事务的互联网应用，MongoDB 完全可以满足 Web 2.0 和移动互联网的数据存储需求。

1.2　MySQL 安装与配置

MySQL 支持多个平台，不同平台下的安装和配置的过程也不相同。考虑到初学者习惯使用 Windows 系统，本节讲解如何在 Windows 平台中安装与配置 MySQL。

1.2.1　获取 MySQL

打开 MySQL 的官方网站 https://www.mysql.com 获取软件的下载。在网站中找到 DOWNLOADS 下载页面，可以看到 MySQL 各种版本的下载地址，如图 1-19 所示。

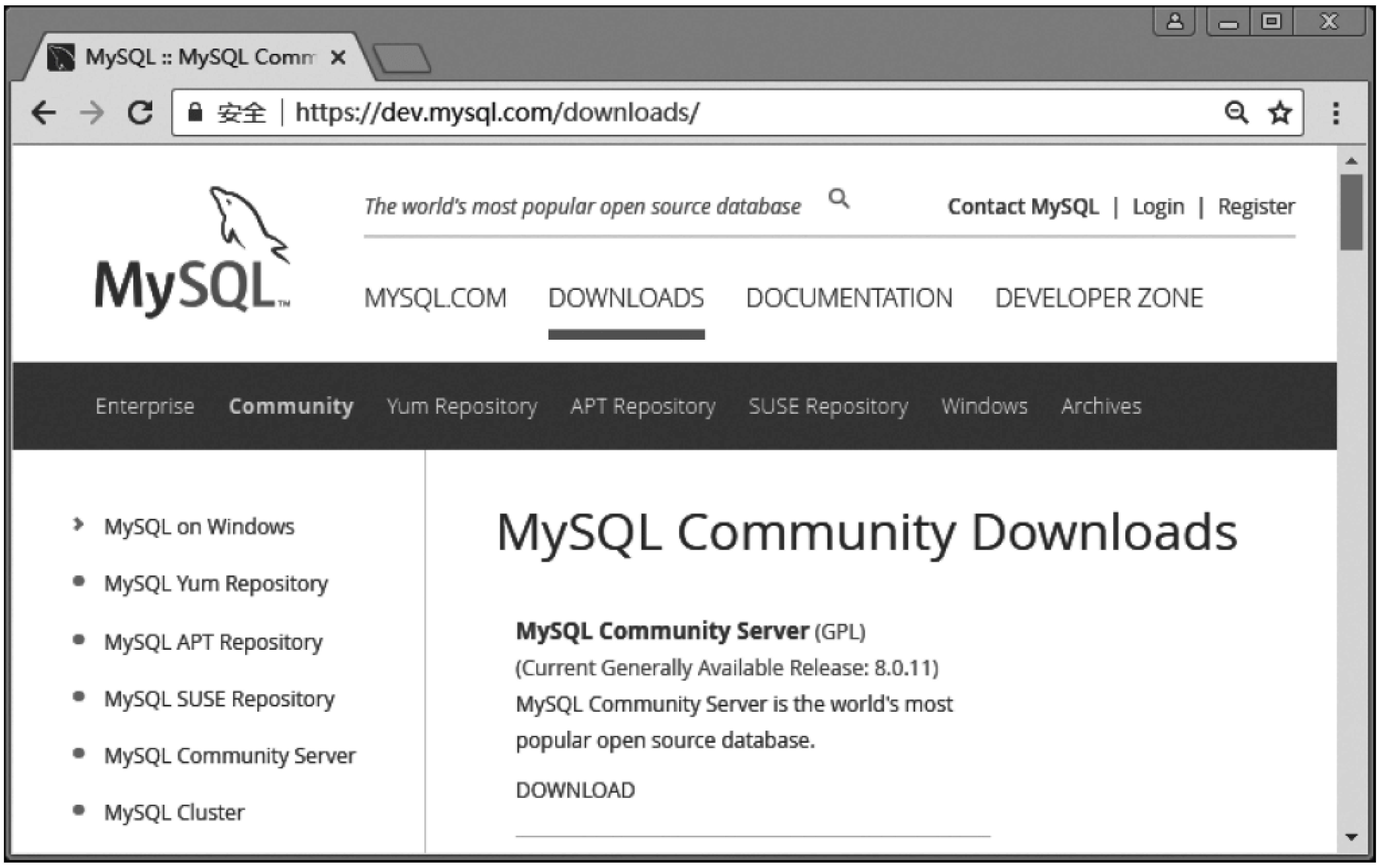


图 1-19　获取 MySQL

在下载页面，MySQL 主要提供了企业版(Enterprise)和社区版(Community)产品，其中社区版是通过 GPL 协议授权的开源软件，可以免费使用，而企业版是需要收费的商业软件。本书选择 MySQL 社区版进行讲解。

在图 1-19 所示的页面中单击 MySQL Community Server 版本的 DOWNLOADS 链接，

进入该版本的下载页面，如图 1-20 所示。

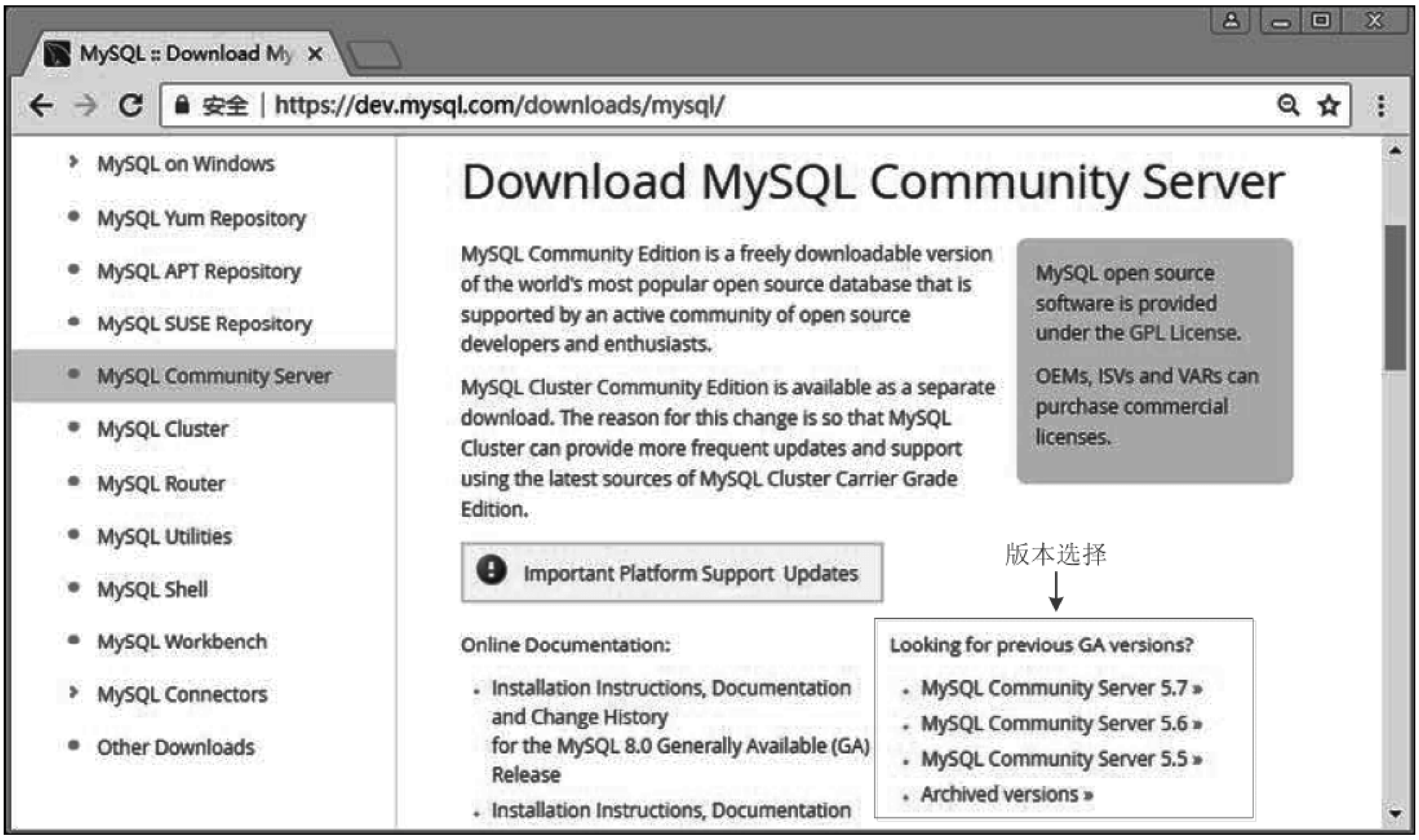


图 1-20　MySQL Community Server 下载页面

在图 1-20 中可以看到，目前发布的最新版本是 8.0，页面中提供了 5.7、5.6 和 5.5 的链接地址。本书选择以 5.7 版本为例进行讲解，进入 MySQL Community Server 5.7 版本的下载页面，如图 1-21 所示。

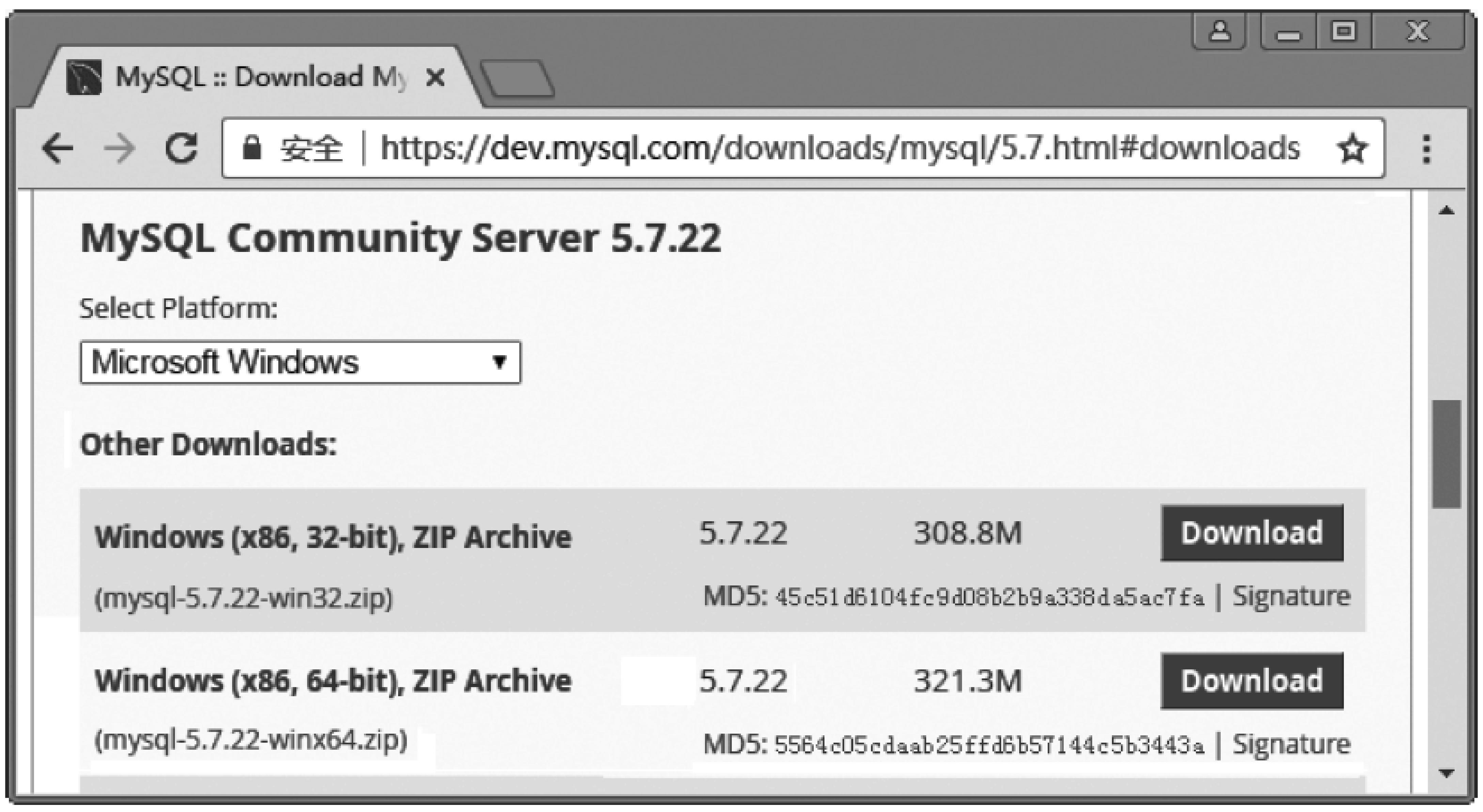


图 1-21　下载 MySQL Community Server 5.7

需要注意的是，MySQL 提供了 32 位和 64 位两种版本，本书以 32 位版本为例进行讲解，在图 1-21 中选择 mysql-5.7.22-win32.zip 进行下载。

小提示：本书讲解的安装方式在 5.5.60、5.6.40、5.7.22、8.0.11 版本中测试通过，这 4 种版本的安装方式并没有太大的差别。

1.2.2　安装 MySQL

1. 解压文件

首先创建 C:\mysql5.7 作为 MySQL 的安装目录，然后打开 mysql-5.7.22-win32.zip 压缩包，将里面的 mysql-5.7.22-win32 目录中的文件解压到 C:\mysql5.7 目录，如图 1-22 所示。

接下来对图 1-22 中的文件分别进行介绍。

(1) bin 目录：用于存放一些可执行文件，如 MySQL 服务程序 mysqld.exe、命令行客户端工具 mysql.exe 等。

(2) docs 目录：用于存放一些文档，如 ChangeLog。

(3) include 目录：用于存放一些头文件，如 mysql.h、mysql_version.h 等。

(4) lib 目录：用于存放一系列的库文件。

(5) share 目录：用于存放字符集、语言等信息。

(6) COPYING 文件：GPL 协议内容。

(7) README 文件：介绍了版权、版本等信息。

2. 安装 MySQL

安装 MySQL 是指将 MySQL 安装为 Windows 系统的服务，具体步骤如下。

(1) 执行【开始】菜单→【所有程序】→【附件】，找到【命令提示符】并右击，在弹出的快捷菜单中选择【以管理员身份运行】方式，启动命令行窗口。

(2) 在命令模式下，切换到 MySQL 安装目录下的 bin 目录。

```
cd C:\mysql5.7\bin
```

(3) 输入以下命令开始安装。

```
mysqld - install
```

安装成功的效果如图 1-23 所示。



图 1-22　MySQL 安装目录



图 1-23　通过命令行安装 MySQL

小提示：在计算机中，服务是一种长时间运行的应用程序，多个服务用不同的端口来区

分。以生活中常见的银行柜台为例，有许多窗口提供服务，多个窗口之间通过编号进行区分，并且这些窗台需要有人值班，来为顾客提供服务。计算机中的服务就相当于银行柜台的窗口，服务的端口号就相当于柜台窗口的编号，而计算机中服务的运行就相当于柜台窗口人员的"值班"。

在安装 MySQL 时，还有一些常见的问题需要注意，具体如下。

（1）MySQL 安装的服务名默认为"MySQL"，如果该名称已经存在，则会安装失败，提示 The service already exists!。此时可能是系统中已经安装了 MySQL，可以通过如下命令进行卸载，卸载后再进行安装。

```
mysqld –remove
```

（2）MySQL 允许在安装或卸载时指定服务名称，从而实现多个 MySQL 服务共存，命令如下所示。

```
mysqld –install "服务名称"
mysqld –remove "服务名称"
```

例如，当需要同时安装 MySQL 5.7 和 8.0 时，分别指定不同的服务名称即可实现。

（3）MySQL 服务默认监听 3306 端口，如果该端口被其他服务占用，会导致客户端无法连接服务器。在命令行中可用 netstat -ano 命令查看端口占用情况，如图 1-24 所示。



图 1-24　查看端口占用情况

从图 1-24 中可以看出，PID 为 4204 的进程正在监听本地地址的 3306 端口，为了获知该进程是哪一个程序，执行 tasklist | findstr "4204"命令，如图 1-25 所示。

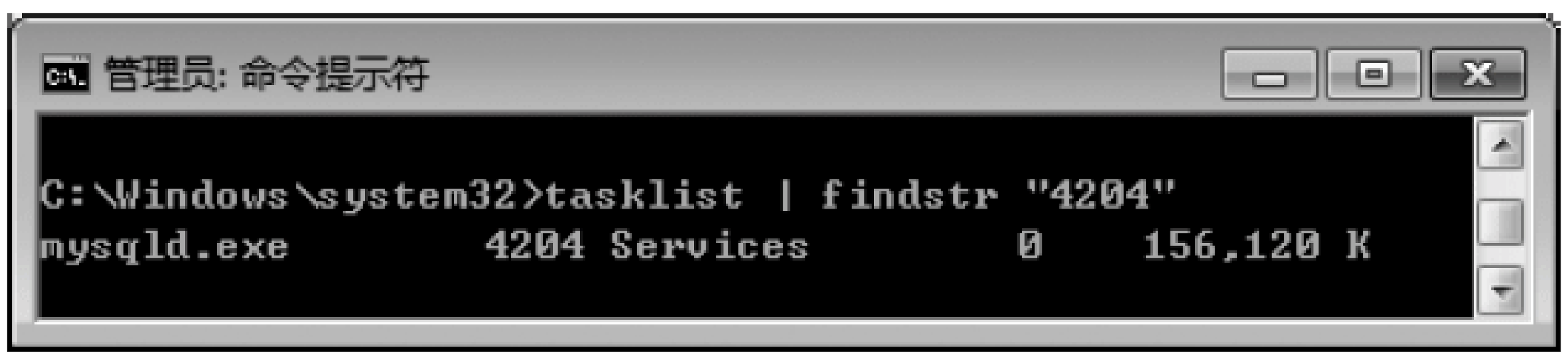


图 1-25　查看进程 ID 对应的程序名称

从图 1-25 中可以看出，当前是 mysqld.exe 占用了 3306 端口，说明 MySQL 服务正在工作。如果是其他程序占用了 3306 端口，只需将对应的服务停止即可。

1.2.3　配置 MySQL

1. 创建 MySQL 配置文件

使用文本编辑器(如记事本、Notepad++ 等工具)创建配置文件 C:\mysql5.7\my.ini,在配置文件中编写如下配置。

```
[mysqld]
basedir=C:/mysql5.7
datadir=C:/mysql5.7/data
port=3306
```

在上述配置中,basedir 表示 MySQL 的安装目录,datadir 表示数据库文件的保存目录,port 表示 MySQL 服务的端口号。

小提示: 在没有配置文件的情况下,MySQL 会自动检测安装目录、数据文件目录。但由于不同 MySQL 版本的路径可能有区别,所以建议通过配置文件来指定。另外,Linux 系统中通常使用 my.cnf 作为配置文件的文件名,在 Windows 系统中也可以使用该文件名。

2. 初始化数据库

创建 my.ini 配置文件后,数据库文件目录 C:\mysql5.7\data 还没有创建。接下来需要通过 MySQL 的初始化功能,自动创建数据文件目录,具体命令如下。

```
mysqld --initialize-insecure
```

在上述命令中,"--initialize"表示初始化数据库,"-insecure"表示忽略安全性。当省略"-insecure"时,MySQL 将自动为默认用户"root"生成一个随机的复杂密码,而加上"-insecure"时,"root"用户的密码为空。由于自动生成的密码输入比较麻烦,因此这里选择忽略安全性。关于密码的设置会在后面进行具体讲解。

小提示: MySQL 5.5 和 5.6 版本中已经提供了 data 目录,不需要初始化数据库。只有安装 5.7 和 8.0 版本时需要执行上述命令。

1.2.4　管理 MySQL 服务

MySQL 安装完成后,需要启动服务进程,否则客户端无法连接数据库。在前面的配置过程中,已经将 MySQL 安装为 Windows 服务。为了控制 MySQL 服务的启动与停止,可以通过两种方式来实现。

1. 通过命令行管理 MySQL 服务

MySQL 服务不仅可以通过 Windows 服务管理器启动,还可以通过命令行来启动。使用管理员身份打开命令提示符,输入如下命令启动名称为 MySQL 的服务。

```
net start MySQL
```

执行完上述命令，显示的结果如图 1-26 所示。

通过命令行不仅可以启动 MySQL 服务，还可以停止 MySQL 服务，具体命令如下。

```
net stop MySQL
```

执行完上述命令，显示的结果如图 1-27 所示。



图 1-26 启动 MySQL 服务



图 1-27 停止 MySQL 服务

2. 通过 Windows 服务管理器管理 MySQL 服务

通过 Windows 的服务管理器可以查看 MySQL 服务是否开启，在命令提示符中输入 services. msc 命令，就会打开 Windows 的服务管理器，如图 1-28 所示。

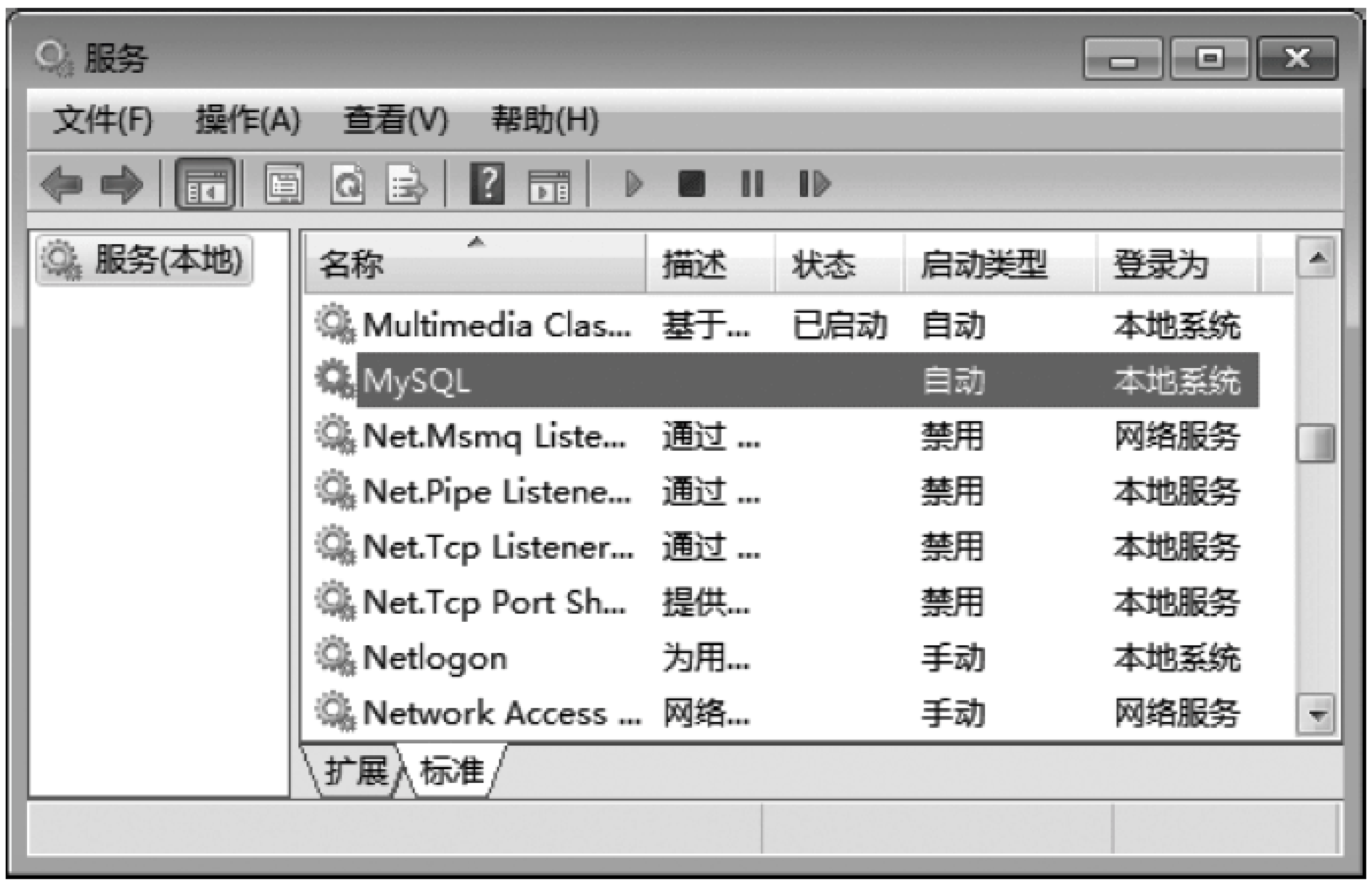


图 1-28 Windows 服务管理器

从图 1-28 可以看出，MySQL 服务没有启动，此时可以直接双击 MySQL 服务项打开属性对话框，通过单击“启动”按钮修改服务的状态，如图 1-29 所示。

图 1-29 中有一个启动类型的选项，该选项有 3 种类型可供选择，具体如下。

(1) 自动：通常与系统有紧密关联的服务才必须设置为自动，它会随系统一起启动。

(2) 手动：服务不会随系统一起启动，直到需要时才会被激活。

(3) 禁用：服务将不能启动。

针对上述 3 种情况，初学者可以根据实际需求进行选择，在此建议选择“自动”或者“手动”。



图 1-29　MySQL 属性对话框

1.2.5　用户登录与设置密码

1. 登录 MySQL

在 MySQL 的 bin 目录中，mysql.exe 是 MySQL 提供的命令行客户端工具，用于访问数据库。该程序不能直接双击运行，需要打开命令行窗口，执行 cd C:\mysql5.7\bin 命令切换工作目录，然后执行如下命令登录 MySQL 服务器。

```
mysql -u root
```

在上述命令中，“mysql”表示运行当前目录下的 mysql.exe；“-u root”表示以 root 用户的身份登录，其中，“-u”和“root”之间的空格可以省略。

成功登录 MySQL 服务器后，运行效果如图 1-30 所示。

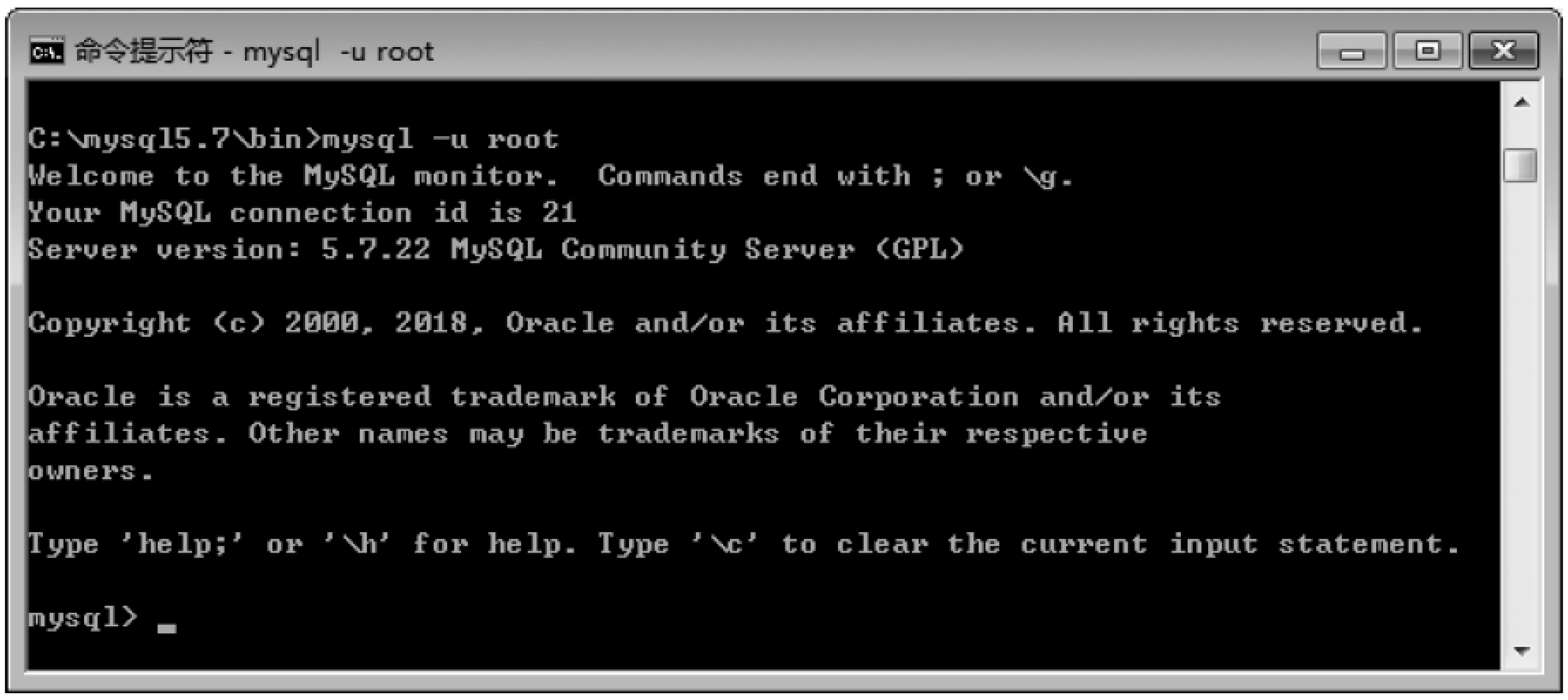


图 1-30　登录 MySQL 数据库

如果需要退出 MySQL，可以直接使用 exit 或 quit 命令。

小提示：命令行客户端工具还有一些常用选项。其中，“-h”用于指定登录的 MySQL 服务器地址(域名或 IP)，如“-h localhost”或“-h 127.0.0.1”表示登录本地服务器。选项“-P”(必须用大写字母 P)用于指定连接的端口号，如“—P 3306”表示连接 3306 端口。

2. 设置密码

为了保护数据库的安全，需要为登录 MySQL 服务器的用户设置密码。下面以设置 root 用户的密码为例，登录 MySQL 后，执行如下命令即可。

```
mysql>ALTER USER 'root'@'localhost' IDENTIFIED BY '123456';
```

上述命令表示为 localhost 主机中的 root 用户设置密码，密码为“123456”。当设置密码后，退出 MySQL，然后重新登录时，就需要输入刚才设置的密码。

在登录有密码的用户时，需要使用的命令如下。

```
mysql -uroot -p123456
```

在上述命令中，“-p123456”表示使用密码“123456”进行登录。如果在登录时不希望密码被直接看到，可以省略“-p”后面的密码，然后按回车键，会提示输入密码。

在设置密码后，如果需要取消密码，可以使用如下命令。

```
mysql>ALTER USER 'root'@'localhost' IDENTIFIED BY '';
```

上述命令将密码设为空，即可免密码登录。

多学一招：设置环境变量

在启动 MySQL 客户端前，需要确保命令提示符当前位于 C:\mysql5.7\bin 目录，如果在其他目录，则需要通过 cd 命令切换目录。这样操作比较麻烦，可以在命令行中执行如下命令，将 MySQL 的 bin 目录添加到环境变量中。

```
setx PATH "%PATH%;C:\mysql5.7\bin"
```

执行上述命令后，关闭当前命令行窗口，重新打开一个新的命令行窗口即可生效。

1.2.6 MySQL 客户端的相关命令

对于初学者来说，使用命令行客户端工具登录 MySQL 数据库后，还不知道如何进行操作。为此，可以查看帮助信息，在命令行中输入 help 或者\h 命令，就会显示 MySQL 客户端的帮助信息，如图 1-31 所示。

图 1-31 中列出了 MySQL 的命令，这些命令既可以使用一个单词来表示，也可以通过“\字母”的方式来表示。为了让初学者更好地掌握 MySQL 相关命令，接下来通过表 1-2 列举 MySQL 中的常用命令。

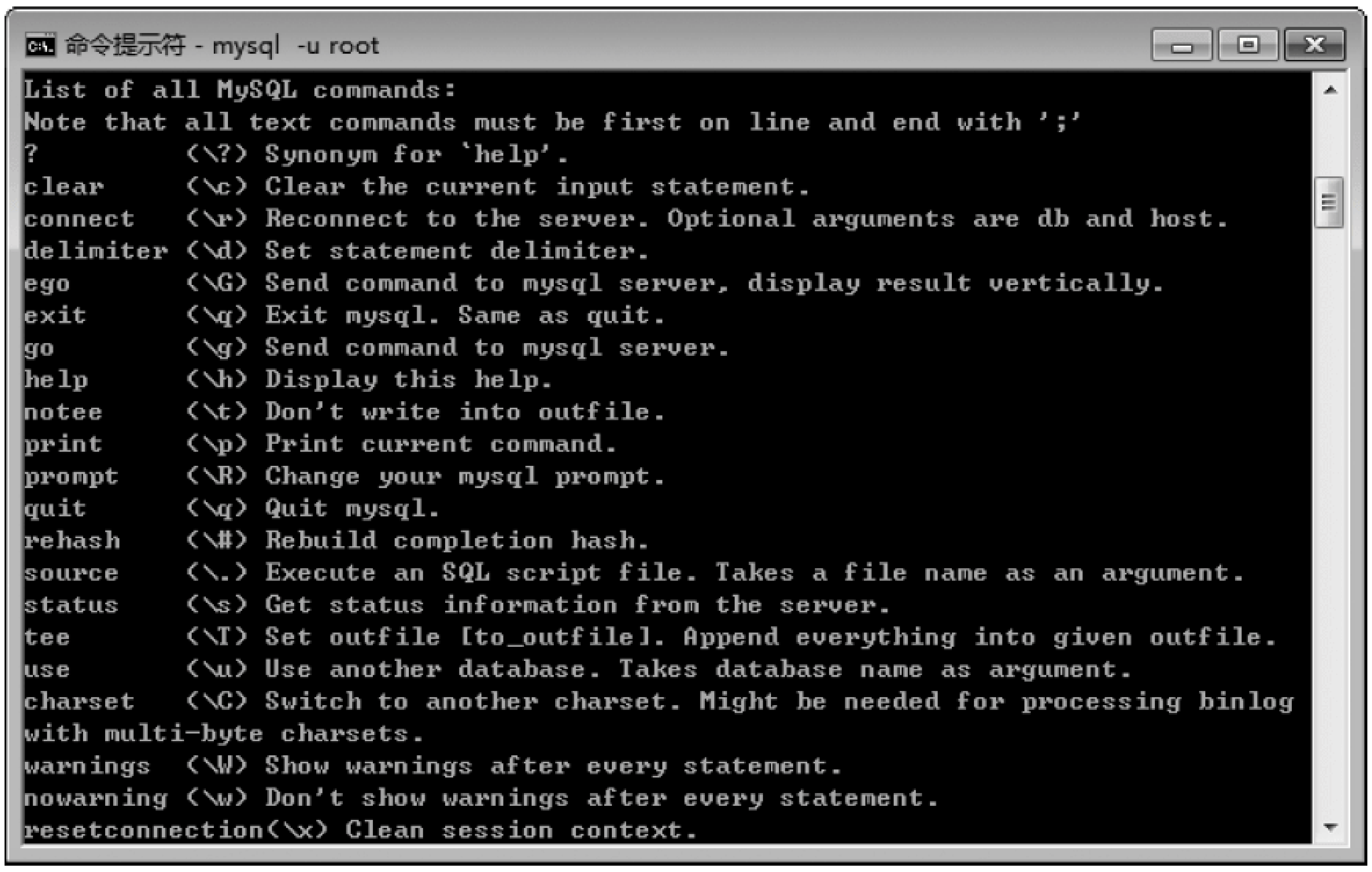


图 1-31　MySQL 相关命令

表 1-2　MySQL 相关命令

命　令	简　写	具体含义
?	\?	显示帮助信息
clear	\c	清除当前输入语句
connect	\r	连接到服务器，可选参数为数据库和主机
delimiter	\d	设置语句分隔符
ego	\G	发送命令到 MySQL 服务器，并显示结果
exit	\q	退出 MySQL
go	\g	发送命令到 MySQL 服务器
help	\h	显示帮助信息
notee	\t	不能将数据导出到文件中
print	\p	打印当前命令
prompt	\R	改变 MySQL 提示信息
quit	\q	退出 MySQL
rehash	\#	重建完成散列，用于表名自动补全
source	\.	执行一个 SQL 脚本文件，以一个文件名作为参数
status	\s	从服务器获取 MySQL 的状态信息
tee	\T	设置输出文件，将所有信息添加到给定的输出文件中
use	\u	选择一个数据库使用，参数为数据库名称
charset	\C	切换到另一个字符集

续表

命　　令	简　　写	具体含义
warnings	\W	每一个语句之后显示警告
nowarnings	\w	每一个语句之后不显示警告
resetconnection	\x	清理会话上下文信息

接下来演示使用 status 命令查看 MySQL 服务器状态信息，结果如下。

```
mysql> status
--------------
mysql  Ver 14.14 Distrib 5.7.22, for Win32 (AMD64)

Connection id:          2
Current database:
Current user:           root@localhost
SSL:                    Not in use
Using delimiter:        ;
Server version:         5.7.22 MySQL Community Server (GPL)
Protocol version:       10
Connection:             localhost via TCP/IP
Server characterset:    latin1
Db     characterset:    latin1
Client characterset:    gbk
Conn. characterset:     gbk
TCP port:               3306
Uptime                  :18 min 26 sec

Threads: 1 Questions: 5 Slow queries: 0 Opens: 105 Flush tables: 1 Open tab
les: 98  Queries per second avg: 0.006
--------------
```

从上述信息可以看出，使用 status 命令显示了 MySQL 当前的版本，字符集以及端口号等信息。关于字符集的内容将会在后面的章节中详细讲解。

1.3 常用图形化工具

MySQL 命令行客户端的优点在于不需要额外安装，在 MySQL 软件包中已经提供。然而命令行这种操作方式不够直观，而且容易出错。为了更方便地操作 MySQL，可以使用一些图形化工具。本节将对 MySQL 常用的两种图形化工具进行讲解。

1.3.1 SQLyog

SQLyog 是 Webyog 公司推出的一个快速、简洁的图形化工具，用于管理 MySQL 数据库。该软件提供了个人版、企业版等版本，并发布了 GPL 协议开源的社区版。

下面以 SQLyog Community Edition-13.0.1(32-Bit)版本为例演示，软件的主界面如图 1-32 所示。



图 1-32　SQLyog 主界面

执行菜单栏【文件】→【新连接】，会弹出如图 1-33 所示的对话框。



图 1-33　连接数据库

输入正确的 MySQL 主机地址（MySQL Host Address）、用户名、密码和端口后，单击【连接】按钮，即可连接数据库。连接成功后的结果如图 1-34 所示。



图 1-34　SQLyog 主界面

在图 1-34 中，左边栏是一个树形控件，root@localhost 表示当前使用 root 用户身份登

录了 localhost 地址的 MySQL 服务器。该服务器中有 4 个数据库，每个都有特定用途，建议初学者不要对这些数据库进行更改操作。

单击每个数据库名称前面的＋按钮，可以查看数据库的内容，如表、视图、存储过程、函数、触发器、事件等。关于这些内容会在后面的章节中详细讲解。

在“询问”面板中可以输入 SQL 语句，输入完成后，单击工具栏中的第 3 个按钮⊙执行查询。

1.3.2 Navicat

Navicat 是一套快速、可靠的图形化数据库管理工具，它的设计符合数据库管理员、开发人员及中小企业的需要。支持的数据库包括 MySQL、MariaDB、SQL Server、SQLite、Oracle 以及 PostgreSQL。

下面以 Navicat 12 版本为例演示。打开软件后，在菜单栏执行【文件】→【新建连接】→【MySQL】，打开新建连接对话框，如图 1-35 所示。



图 1-35 新建连接

在图 1-35 中，输入连接名(如“新连接”)、主机名或 IP 地址、端口、用户名和密码后，单击【确定】按钮，即可连接数据库。连接成功后的结果如图 1-36 所示。

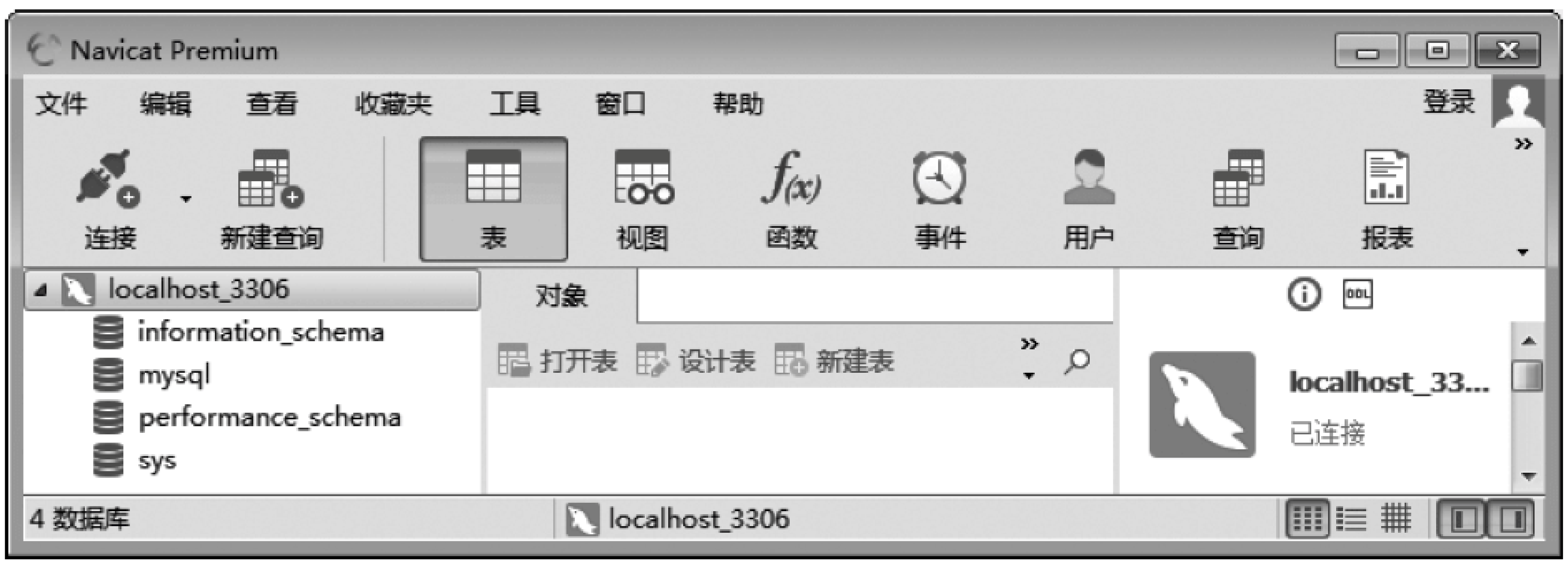


图 1-36 Navicat 主界面

单击工具栏中的【新建查询】按钮，可以执行 SQL，如图 1-37 所示。



图 1-37　新建查询

1.4　本章小结

本章主要讲解了数据库的基础知识、MySQL 的安装与配置以及 MySQL 的使用。通过本章的学习，希望初学者真正掌握 MySQL 数据库的基础知识，并且学会在 Windows 平台上安装与配置 MySQL，为后面章节的学习奠定扎实的基础。

1.5　课后练习

一、填空题

1. 关系数据库的标准语言是________。
2. 数据库发展的 3 个阶段中，数据独立性最高的是________阶段。
3. 概念模型中的 3 种基本联系分别是________、________和________。
4. MySQL 配置文件的文件名是________。
5. 在 MySQL 配置文件中，________用于指定数据库文件的保存目录。

二、判断题

1. 数据只包括普通意义上的数字和文字。(　　)
2. 关系模型的数据结构是二维表。(　　)
3. 概念模式是对数据库的整体逻辑结构的描述。(　　)
4. 数据冗余度高是数据库系统的特点之一。(　　)
5. SQL 是指标准化查询语言。(　　)

三、选择题

1. 三级模式是对(　　)的三个抽象级别。
   A. 数据库系统　B. 数据库　C. 数据　D. 数据库管理系统
2. 数据的独立性包括(　　)。
   A. 物理独立性　B. 逻辑独立性　C. 用户独立性　D. 程序独立性
3. 数据库的完整性是指数据的(　　)。
   A. 正确性　B. 一致性　C. 相容性　D. 安全性
4. 下列选项中,不属于 DDL 语句的是(　　)。
   A. CREATE 语句　B. ALTER 语句
   C. DROP 语句　D. SELECT 语句
5. 下列选项中,哪个是 MySQL 默认提供的用户(　　)。
   A. admin　B. test　C. root　D. user

四、简答题

1. 请简述什么是数据模型。
2. 请简述数据库、表和数据库服务器之间的关系。

第 2 章
数据库基本操作

学习目标

- 掌握数据库的创建、查看、选择与删除操作
- 掌握数据表的创建、查看、修改与删除操作
- 掌握数据的添加、查询、修改与删除操作

在 MySQL 数据库的学习中，数据库、数据表和数据的操作，是每个初学者必须掌握的内容，同时也是学习后续课程的基础。为了让初学者能够快速体验与掌握数据库的基本操作，本章将对这些基本操作进行详细讲解。

2.1 数据库操作

2.1.1 创建数据库

MySQL 服务器中的数据库可以有多个，分别存储不同的数据。要想将数据存储到数据库中，首先需要创建一个数据库。创建数据库就是在数据库系统中划分一块存储数据的空间，基本语法格式如下。

```
CREATE DATABASE 数据库名称 [库选项];
```

在上述语法中，CREATE DATABASE 表示创建数据库；“数据库名称”可以是字母、数字和下划线组成的任意字符串；“库选项”用于设置此数据库的相关特性，如字符集 CHARSET，校对集 COLLATE。

其中，语法内使用“[]”括起来的选项表示可选参数。具体库选项的设置及相关注意事项会在后面的章节中讲解，此处读者了解即可。

下面创建一个名称为 mydb 的数据库，具体 SQL 语句与执行结果如下。

```
mysql>CREATE DATABASE mydb;
Query OK, 1 row affected (0.00 sec)
```

在创建数据库后，MySQL 会在存储数据的 data 目录中创建一个与数据库同名的子目录（即 mydb），同时会在 mydb 目录下生成一个 db.opt 文件，保存数据库选项。打开 data\mydb\db.opt 文件，如下所示。

```
default-character-set=latin1
default-collation=latin1_swedish_ci
```

上述内容表示mydb数据库的默认字符集为latin1，校对集为latin1_swedish_ci。

值得一提的是，如果创建的数据库已存在，则程序会报错。为了防止这种情况的发生，在创建数据库时可以在“数据库名称”前添加IF NOT EXISTS，表示指定的数据库不存在时执行创建操作，否则忽略此操作。

例如，再次创建一个名称为mydb的数据库，具体SQL语句如下。

```
mysql>CREATE DATABASE IF NOT EXISTS mydb;
Query OK, 1 row affected, 1 warning (0.00 sec)
```

从以上结果可以看出，创建数据库时添加IF NOT EXISTS后，再次创建mydb就不会发生错误，但是服务器返回了一条警告信息。下面可通过SHOW WARNINGS查看错误信息，具体SQL语句如下。

```
mysql>SHOW WARNINGS;
+--------+------+------------------------------------------------+
| Level  | Code | Message                                        |
+--------+------+------------------------------------------------+
| Note   | 1007 | Can't create database 'mydb'; database exists  |
+--------+------+------------------------------------------------+
1 row in set (0.00 sec)
```

从以上结果可知，MySQL提示名为mydb的数据库已经存在，不能再重复创建。

小提示：从前面的SQL语句操作可以看出，创建数据库就是在存储数据的文件夹data下生成一个与数据库同名的目录，用于保存此数据库相关的内容。因此，在MySQL中还可以通过在data下创建目录的方式完成数据库的创建。

2.1.2 查看数据库

数据库创建完成后，若要查看该数据库的信息，或查看MySQL服务器当前都有哪些数据库，可以根据不同的需求选择以下的方式进行查看。

1. 查看MySQL服务器下所有数据库

当需要查看MySQL服务器中已经存在的数据库时，基本语法格式如下。

```
SHOW DATABASES;
```

执行上述SQL语句后，运行效果如下所示。

```
mysql>SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
```

```
| information_schema       |
| mysql                    |
| performance_schema       |
| mydb                     |
| sys                      |
+--------------------+-
5 rows in set (0.00 sec)
```

在以上输出的结果中，MySQL 服务器已有 5 个数据库，除 mydb 是手动创建的数据库外，其他数据库都是 MySQL 安装时自动创建的。

information_schema 和 performance_schema 数据库分别是 MySQL 服务器的数据字典（保存所有数据表和库的结构信息）和性能字典（保存全局变量等的设置）；“mysql”数据库主要负责 MySQL 服务器自己需要使用的控制和管理信息，如用户的权限关系等；sys 是系统数据库，包括了存储过程、自定义函数等信息。对于初学者来说，建议不要随意的删除和修改这些数据库，避免造成服务器故障。

2. 查看指定数据库的创建信息

在完成创建数据库后，若要查看创建该数据库的信息，基本语法格式如下。

```
SHOW CREATE DATABASE 数据库名称；
```

接下来查看前面创建的数据库 mydb，具体 SQL 语句与执行结果如下。

```
mysql> SHOW CREATE DATABASE mydb;
+----------+-------------------------------------------------------------+
| Database | Create Database                                             |
+----------+-------------------------------------------------------------+
|  mydb    | CREATE DATABASE `mydb`     /* ! 40100 DEFAULT CHARACTER SET latin1 * /   |
+----------+-------------------------------------------------------------+
1 row in set (0.00 sec)
```

以上输出结果显示了创建 mydb 数据库的 SQL 语句，以及数据库的默认字符集。

2.1.3　选择数据库

由于 MySQL 服务器中的数据需要存储到数据表中，而数据表需要存储到对应的数据库下，并且 MySQL 服务器中又可以同时存在多个数据库，因此，在对数据和数据表进行操作前，首先需要选择数据库。基本语法格式如下。

```
USE 数据库名称；
```

接下来选择数据库 mydb 进行操作，具体 SQL 语句与执行结果如下。

```
mysql> USE mydb;
Database changed
```

多学一招：登录 MySQL 服务器时选择数据库

数据库的选择除了可以使用 USE 关键字外，在用户登录 MySQL 服务器时也可以直接选择要操作的数据库，基本语法格式如下。

```
mysql -u用户名 -p密码 数据库名
```

上述语法中，在用户登录服务器的密码后添加要选择的数据库名称，按回车键后，MySQL 会在登录服务器后自动选择要操作的数据库。例如，密码为 123456 的 root 用户登录后要直接选择 mydb 数据库进行操作，具体 SQL 语句如下。

```
#方式 1,在登录时显示用户密码,选择数据库
mysql -uroot -p123456 mydb
#方式 2,在登录时隐藏用户密码,选择数据库
mysql -uroot -p mydb
Enter password: ******
```

2.1.4 删除数据库

在 MySQL 中如若要清除数据库中的所有数据，回收为数据库分配的存储空间，则可以执行删除数据库的操作，基本语法格式如下。

```
DROP DATABASE 数据库名称;
```

在上述语法中，DROP DATABASE 表示删除数据库，“数据库名称”是待删除的数据库名称。下面以删除一个名称为 mydb 的数据库为例进行演示，具体 SQL 语句与执行结果如下。

```
mysql> DROP DATABASE mydb;
Query OK, 0 rows affected (0.01 sec)
```

需要注意的是，在使用 DROP DATABASE 删除数据库时，若待删除数据库(如 mydb)不存在，MySQL 服务器会报错。因此，可以在删除数据库时，使用 IF EXISTS，具体 SQL 语句与执行结果如下。

```
mysql> DROP DATABASE IF EXISTS mydb;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

上述 SQL 语句表示，若 MySQL 服务器中存在数据库 mydb，则删除该数据库，否则不执行删除数据库 mydb 的操作。这与创建一个已存在的数据库相同，MySQL 服务器也会返回一条警告信息用于提示，读者可通过 SHOW WARNINGS 查看，这里不再演示。

值得一提的是，在执行删除数据库操作前，一定要备份需要保留的数据，确保数据的安全，避免误操作造成严重的后果。

多学一招：注释

MySQL 中的注释通常可以分为两类：一类是将注释内容添加到表结构中(此处了解即

可，具体会在创建表时详细讲解)；另一类则会在服务器实际运行时被忽略，有单行注释和多行注释之分。下面对这两种注释详细讲解。

MySQL 中单行注释以"#"开始标识，也支持标准 SQL 中"-- "单行注释。但是为了防止"-- "与 SQL 语句中负号和减法运算的混淆，在第二个短横线后必须添加至少一个控制字符(如空格、制表符、换行符等)将其标识为单行注释符号。示例如下。

```
#此处填写单行注释内容，如：若服务器中没有 mydb 数据库，则创建，否则忽略此 SQL
CREATE DATABASE IF NOT EXISTS mydb;
-- 此处填写单行注释内容，如：若服务器中存在 mydb 数据库，则删除，否则忽略此 SQL
DROP DATABASE IF EXISTS mydb;
```

同样地，MySQL 也支持标准 SQL 中的多行注释"/* 此处填写注释内容 */"，它的开始符号为"/*"，结束符号为"*/"，中间的内容就是要编写的注释。示例如下。

```
/*
  此处填写多行注释内容
  如：利用以下 SQL 查看当前服务器中的所有数据库
*/
SHOW DATABASES;
```

在开发中编写的 SQL 语句，建议合理地添加单行或多行注释，方便阅读与理解。

脚下留心

在 MySQL 使用的过程中，它相关的基本语法有以下 3 点需要注意的地方。

(1) 换行、缩进与结尾分隔符。MySQL 中的 SQL 语句可以单行或多行书写，多行书写时可以按回车键换行，每行中的 SQL 语句可以使用空格和缩进增强语句的可读性，在 SQL 语句完成时通常情况下使用分号(;)结尾，在命令行窗口中也可使用"\g"结尾，效果与分号相同。另外，在命令行窗口中，还可以使用"\G"结尾，如 SHOW DATABASES\G 将显示结果以每条记录(一行数据)为一组，将所有的字段纵向排列展示。

(2) 大小写问题。MySQL 的关键字在使用时不区分大小写，如 SHOW DATABASES 与 show databases 都表示获取当前 MySQL 服务器中有哪些数据库。另外，MySQL 中的所有数据库名称、数据表名称、字段名称默认情况下在 Windows 系统下都忽略大小写，在 Linux 系统下数据库与数据表名称则区分大小写，通常开发时推荐都使用小写。

本书为了读者便于理解，所有 MySQL 关键字均采用大写形式出现，其他自定义的名称(如数据库名)均以小写形式出现。

(3) 反引号的使用。在项目开发中，为了避免用户自定义的名称与系统中的命令(如关键字)冲突，最好使用反引号(``)包裹数据库名称、字段名称和数据表名称。其中，反引号(`)在键盘中左上角 Tab 键的上方，读者只需将输入法切换到英文，按下此键即可输入反引号(`)。

2.2　数据表操作

在 MySQL 数据库中，所有的数据都存储在数据表中，若要对数据执行添加、查看、修改、删除等操作，首先需要在指定的数据库中准备一张数据表。下面将详细地讲解如何在

MySQL中创建、查看、修改以及删除数据表。

2.2.1 创建数据表

创建数据表指的是在已存在的数据库中建立新表。MySQL既可以根据开发需求创建新的表，又可以根据已有的表复制相同的表结构。其中依据已有的表创建相同结构的新表方式会在后面的章节中讲解，此处仅讲解如何根据需求创建一个简单的新表。

在MySQL数据库中，使用CREATE TABLE语句可以完成数据表的创建，基本语法格式如下。

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] 表名
(字段名 字段类型 [字段属性]…) [表选项]
```

在上述语法中，可选项TEMPORARY表示临时表，仅在当前会话中可见，并且在会话关闭时自动删除。“字段名”指的是数据表的列名；“字段类型”设置字段中保存的数据类型，如时间日期类型等；可选项“字段属性”指的是字段的某些特殊约束条件。可选的“表选项”用于设置表的相关特性，如存储引擎（ENGINE）、字符集（CHARSET）和校对集（COLLATE）。

其中，字段类型、字段属性以及表选项的设置和相关注意事项会在后面的章节中讲解，此处读者了解即可。

需要注意的是，在操作数据表之前，应该使用“USE 数据库名”指定操作是在哪个数据库中进行，否则会抛出No database selected错误。

下面在mydb数据库中，创建一个名称为goods的数据表，保存商品信息，具体SQL语句及执行结果如下。

```
#① 创建 mydb 数据库
mysql>CREATE DATABASE mydb;
Query OK, 1 row affected (0.00 sec)
#② 选择 mydb 数据库
mysql>USE mydb;
Database changed
#③ 创建 goods 数据表
mysql>CREATE TABLE goods (
    ->    id INT COMMENT '编号',
    ->    name VARCHAR(32) COMMENT '商品名',
    ->    price INT COMMENT '价格',
    ->    description VARCHAR(255) COMMENT '商品描述'
    ->);
Query OK, 0 rows affected (0.01 sec)
```

上述SQL语句中，INT用于设置字段数据类型是整型；VARCHAR(L)表示可变长度的字符串，L表示字符数，如VARCHAR(32)表示可变的字符数是32；COMMENT用于在创建表时添加注释内容，并将其保存到表结构中。

值得一提的是，在操作数据表时，可以不使用“USE 选择数据库”的方式选择数据库，直接将表名的位置改为“数据库.表名”的形式，就可以在任何数据库下访问其他数据库中的

表。例如，省略以上第②步的操作，将第③步创建 goods 数据表的语句修改成如下形式。

```
CREATE TABLE mydb.goods (此处省略字段的定义);
```

上述语句中，mydb. goods 表示 mydb 数据库中的 goods 数据表。

注意：在为表进行命名时，由于项目开发中，同一个数据库可能被多个项目使用，因此为了避免数据表重复，通常为数据表添加前缀用于区分不同的项目。前缀一般选取数据库的前几个字母，并添加一个下划线(_)。例如，mydb_goods 若是一个表名，则“mydb_”就是表前缀。

2.2.2　查看数据表

MySQL 中提供了专门的 SQL 语句，用于查看某数据库中存在的所有数据表、指定模式的数据表或数据表的相关信息。下面分别对其进行详细讲解。

1. 查看数据表

选择数据库后，可以通过 MySQL 提供的 SQL 语句进行查看，基本语法格式如下。

```
SHOW TABLES [LIKE 匹配模式];
```

上述语法中，若不添加可选项“LIKE 匹配模式”，表示查看当前数据库中的所有数据表；若添加则按照“匹配模式”查看数据表。其中，匹配模式符有两种，分别为“%”和“_”。前者表示匹配一个或多个字符，代表任意长度的字符串，长度也可以为 0，后者仅可以匹配一个字符。

为了读者更好地理解，下面以 mydb 数据库中数据表的查询为例进行演示。首先为 mydb 数据库再添加一张数据表 new_goods，方便读者对以下示例的理解。new_goods 表的创建语句如下。

```
mysql> CREATE TABLE new_goods (
    ->   id INT COMMENT '编号',
    ->   name VARCHAR(32) COMMENT '商品名',
    ->   price INT COMMENT '价格',
    ->   description VARCHAR(255) COMMENT '商品描述'
    -> );
Query OK, 0 rows affected (0.01 sec)
```

数据表准备完成后，接下来分别查看 mydb 数据表中的所有数据表和名称中含有 new 的数据表。具体 SQL 语句如下。

```
#① 查看所有数据表
mysql> SHOW TABLES;
+-----------------+
| Tables_in_mydb  |
+-----------------+
| goods           |
+-----------------+
```

```
1 rows in set (0.00 sec)
#② 查看名称中含有 new 的数据表
mysql> SHOW TABLES LIKE '%new%';
+------------------------+
| Tables_in_mydb (%new%) |
+------------------------+
| new_goods              |
+------------------------+
1 row in set (0.00 sec)
```

从以上输出结果可以看出，mydb 数据库中一共有两个数据表，而名字中含有 new 的数据表仅有一个。需要注意的是，LIKE 后的匹配模式必须使用单引号或双引号包裹。

2. 查看数据表的相关信息

除了查看数据库下有哪些数据表外，还可以利用 MySQL 提供的 SQL 语句查看数据表的相关信息，如数据表的名称、存储引擎、创建时间等，基本语法格式如下。

```
SHOW TABLE STATUS [FROM 数据库名] [LIKE 匹配模式];
```

下面查看 mydb 数据库下含有 new 的数据表的详细信息，具体 SQL 语句如下。

```
mysql> SHOW TABLE STATUS FROM mydb LIKE '%new%'\G
*************************** 1. row ***************************
           Name: new_goods
         Engine: InnoDB
        Version: 10
     Row_format: Dynamic
           Rows: 0
 Avg_row_length: 0
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: NULL
    Create_time: 2018-04-26 15:58:05
    Update_time: NULL
     Check_time: NULL
      Collation: latin1_swedish_ci
       Checksum: NULL
 Create_options:
        Comment:
1 row in set (0.00 sec)
```

上述 SQL 语句中，“\G”是 MySQL 客户端可以使用的结束符中的一种，用于将显示结果纵向排列，适合字段非常多的情况。输出结果中含有值的字段含义如表 2-1 所示，其他字段的含义读者可在 MySQL 手册中查看，此处不再赘述。

表 2-1　数据表的相关信息

字段名称	描　　述
Name	数据表的名称
Engine	数据表的存储引擎
Version	数据表的结构文件(如 lib_user_temp.frm)版本号
Row_format	记录的存储格式，Dynamic 表示动态
Data_length	数据文件的长度(MyISAM 存储引擎)或为集群索引分配的内存(InnoDB 存储引擎)，均以字节为单位
Create_time	数据表的创建时间
Collation	数据表的校对集

在表 2-1 中，Row_format 字段的值除 Dynamic 外，还有 Fixed(固定)、Compressed(压缩)、Redundant(冗余)和 Compact(紧凑)。

2.2.3　修改数据表

在实际开发时，若创建的数据表不符合当前项目的开发要求时，可以通过修改数据表来实现。如修改数据表的名称和表选项。下面将分别讲解如何修改数据表。

1. 修改数据表名称

在 MySQL 中，提供了两种修改数据表名称的方式，基本语法格式如下。

```
#语法格式 1
ALTER TABLE 旧表名 RENAME [TO|AS] 新表名;
#语法格式 2
RENAME TABLE 旧表名 1 TO 新表名 1[, 旧表名 2 TO 新表名 2] ...
```

在上述语法中，ALTER TABLE 修改数据表名称时，可以直接使用 RENAME 或在其后添加 TO 或 AS。而 RENAME TABLE 则必须使用 TO，另外此语法可以同时修改多个数据表的名称。

下面使用 RENAME TABLE 将 new_goods 表的名称修改为 my_goods，具体 SQL 语句与执行结果如下。

```
mysql>RENAME TABLE new_goods TO my_goods;
Query OK, 0 rows affected (0.01 sec)
mysql>SHOW TABLES;
+------------+
| Tables_in_mydb |
+------------+-
| goods        |
| my_goods     |
+------------+-
2 rows in set (0.00 sec)
```

执行上述 SQL 语句后，使用 SHOW TABLES；可查看修改后的数据表。

2. 修改表选项

数据表中的表选项字符集、存储引擎以及校对集也可以通过 ALTER TABLE 修改，基本语法格式如下。

```
ALTER TABLE 表名 表选项 [=] 值;
```

下面以修改 my_goods 数据表的字符集为例进行演示，具体 SQL 语句如下。

```
#① 将 my_goods 数据表的字符集改为 utf8
mysql> ALTER TABLE my_goods CHARSET = utf8;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
#② 查看修改结果
mysql> SHOW CREATE TABLE my_goods \G
*************************** 1. row ***************************
       Table: my_goods
Create Table: CREATE TABLE `my_goods` (
  `id` int(11) DEFAULT NULL COMMENT '编号',
  `name` varchar(32) CHARACTER SET latin1 DEFAULT NULL COMMENT '商品名',
  `price` int(11) DEFAULT NULL COMMENT '价格',
  `description` varchar(255) CHARACTER SET latin1 DEFAULT NULL COMMENT '商品描述'
) ENGINE= InnoDB DEFAULT CHARSET= utf8
1 row in set (0.00 sec)
```

在上述 SQL 语句中，第②步使用 SHOW CREATE TABLE my_goods\G 查看表的字符集，该语句会在下一节中详细讲解，此处读者会使用即可。

2.2.4 查看表结构

1. 查看数据表的字段信息

MySQL 提供的 DESCRIBE 语句可以查看数据表中所有字段或指定字段的信息，包括字段名、字段类型等。其中，DESCRIBE 语句可以简写成 DESC。基本语法格式如下。

```
#语法格式 1: 查看所有字段的信息
{ DESCRIBE | DESC } 数据表名;
#语法格式 2: 查看指定字段的信息
{ DESCRIBE | DESC } 数据表名 字段名;
```

下面以查看数据表 my_goods 中的所有字段和指定字段 name 为例演示。具体 SQL 语句及执行结果如下。

```
#① 所有字段
mysql> DESC my_goods;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
```

```
+-------------+--------------+------+-----+---------+-------+
| id          | int(11)      | YES  |     | NULL    |       |
| name        | varchar(32)  | YES  |     | NULL    |       |
| price       | int(11)      | YES  |     | NULL    |       |
| description | varchar(255) | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
#② name 字段
mysql> DESC my_goods name;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| name  | varchar(32) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
1 row in set (0.00 sec)
```

在上述执行结果中,Field 表示字段名称,Type 表示字段的数据类型,Null 表示该字段是否可以为空,Key 表示该字段是否已设置了索引,Default 表示该字段是否有默认值,Extra 表示获取到的与该字段相关的附加信息。

2. 查看数据表的创建语句

若想要查看创建数据表的具体 SQL 语句以及表的字符编码,可以使用以下的 SQL 语句,基本语法格式如下。

```
SHOW CREATE TABLE 表名;
```

接下来查看 my_goods 数据表的创建语句,具体 SQL 语句及执行结果如下。

```
mysql> SHOW CREATE TABLE my_goods\G
*************************** 1. row ***************************
       Table: my_goods
Create Table: CREATE TABLE `my_goods` (
  `id` int(11) DEFAULT NULL COMMENT '编号',
  `name` varchar(32) CHARACTER SET latin1 DEFAULT NULL COMMENT '商品名',
  `price` int(11) DEFAULT NULL COMMENT '价格',
  `description` varchar(255) CHARACTER SET latin1 DEFAULT NULL COMMENT '商品描述'
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

在上述执行结果中,Table 表示查询的表名称,Create Table 表示创建该数据表的 SQL 语句。在 SQL 语句中,包含了字段信息、COMMENT(注释)、ENGINE(存储引擎)以及 DEFAULT CHARSET(字符集)等内容。

3. 查看数据表结构

MySQL 数据库中的 SHOW COLUMNS 语句也可以查看表结构,基本语法格式如下。

```
#语法格式 1
SHOW [FULL] COLUMNS  FROM 数据表名 [FROM 数据库名];
#语法格式 2
SHOW [FULL] COLUMNS  FROM 数据库名.数据表名;
```

在上述语法格式中，可选项 FULL 表示显示详细内容，在不添加的情况下查询结果与 DESC 的结果相同；在添加 FULL 选项时此语句不仅可以查看到 DESC 语句查看的信息，还可以查看到字段的权限、COMMENT 字段的注释信息等。

另外，在 SQL 语句中可以通过"FROM 数据库名"或"数据库名. 数据表名"的方式查看任意数据库下的数据表结构信息。

下面查看 my_goods 数据表结构的详细信息，具体 SQL 语句如下。

```
mysql> SHOW FULL COLUMNS FROM my_goods;
+-------------+--------------+-------------------+------+-----+
| Field       | Type         | Collation         | Null | Key |
+-------------+--------------+-------------------+------+-----+
| id          | int(11)      | NULL              | YES  |     |
| name        | varchar(32)  | latin1_swedish_ci | YES  |     |
| price       | int(11)      | NULL              | YES  |     |
| description | varchar(255) | latin1_swedish_ci | YES  |     |
+-------------+--------------+-------------------+------+-----+
+---------+-------+---------------------------------+----------+
| Default | Extra | Privileges                      | Comment  |
+---------+-------+---------------------------------+----------+
| NULL    |       | select,insert,update,references | 编号     |
| NULL    |       | select,insert,update,references | 商品名   |
| NULL    |       | select,insert,update,references | 价格     |
| NULL    |       | select,insert,update,references | 商品描述 |
+---------+------ +---------------------------------+----------+
4 rows in set (0.00 sec)
```

从上述执行结果可以看出，SHOW FULL COLUMNS 语句除与 DESC 语句查询出的相同字段外，还包括 Collation（校对集）字段、Privileges（权限）字段和 Comment（注释）字段。

2.2.5 修改表结构

在创建完数据表后，除了可以修改数据表的名称及表选项外，还可以利用 MySQL 提供的 ALTER TABLE 语法对字段名称、类型、位置等进行修改、增加或删除。下面分别讲解几种常用的使用方式。

1. 修改字段名

在 MySQL 中仅修改数据表中的字段名称，使用 CHANGE 实现，基本语法格式如下。

```
ALTER TABLE 数据表名 CHANGE [COLUMN] 旧字段名 新字段名 字段类型 [字段属性];
```

在上述语法中，"旧字段名"指的是字段修改前的名称，"新字段名"指的是字段修改后的

名称。“数据类型”表示新字段名的数据类型，不能为空，即使与旧字段的数据类型相同，也必须重新设置。

下面将 my_goods 数据表中名为 description 的字段修改为 des，具体 SQL 语句如下。

```
mysql>ALTER TABLE my_goods CHANGE description des VARCHAR(255);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

执行上述 SQL 语句后，查看字段名的修改情况，具体结果如下。

```
mysql>DESC my_goods;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| id    | int(11)      | YES  |     | NULL    |       |
| name  | varchar(32)  | YES  |     | NULL    |       |
| price | int(11)      | YES  |     | NULL    |       |
| des   | varchar(255) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

2. 修改字段类型

在 MySQL 中仅修改数据表中的字段类型，通常使用 MODIFY 实现，基本语法格式如下。

```
ALTER TABLE 数据表名 MODIFY [COLUMN] 字段名 新类型 [字段属性];
```

下面修改 my_goods 数据表中 des 字段的数据类型，将 VARCHAR（255）修改为 CHAR(255)，具体 SQL 语句如下。

```
mysql>ALTER TABLE my_goods MODIFY des CHAR(255);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

执行上述 SQL 语句后，查看字段类型的修改情况，具体结果如下。

```
mysql>DESC my_goods des;
+-------+-----------+------+-----+---------+-------+
| Field | Type      | Null | Key | Default | Extra |
+-------+-----------+------+-----+---------+-------+
| des   | char(255) | YES  |     | NULL    |       |
+-------+-----------+------+-----+---------+-------+
1 row in set (0.00 sec)
```

3. 修改字段的位置

数据表在创建时，字段编写的先后顺序就是其在数据库中存储的顺序，若需要调整某个

字段的位置，也可以使用 MODIFY 实现，基本语法格式如下。

```
ALTER TABLE 数据表名
MODIFY [COLUMN] 字段名 1  数据类型 [字段属性] [FIRST | AFTER 字段名 2];
```

从上述语法可知，修改字段的位置就是在修改字段类型的后面添加“FIRST”或“AFTER 字段名 2”。前者表示将“字段名 1”调整为数据表的第 1 个字段，后者表示将“字段名 1”插入到“字段名 2”的后面。

下面将 my_goods 表中最后一个字段 des 移动到 name 字段后，具体 SQL 语句如下。

```
mysql> ALTER TABLE my_goods MODIFY des VARCHAR(255) AFTER name;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

执行上述 SQL 语句后，查看字段位置的修改结果，具体结果如下。

```
mysql> DESC my_goods;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| id    | int(11)      | YES  |     | NULL    |       |
| name  | varchar(32)  | YES  |     | NULL    |       |
| des   | varchar(255) | YES  |     | NULL    |       |
| price | int(11)      | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

4. 新增字段

对于已经创建好的数据表，也可以根据业务需求利用 ADD 新增字段，基本语法格式如下。

```
#语法格式 1: 新增一个字段,并可指定其位置
ALTER TABLE 数据表名
ADD [COLUMN] 新字段名 字段类型 [FIRST | AFTER 字段名]
#语法格式 2: 同时新增多个字段
ALTER TABLE 数据表名
ADD [COLUMN] (新字段名 1  字段类型 1, 新字段名 2  字段类型 2, …)
```

在上述语法中，在不指定位置的情况下，新增的字段默认添加到表的最后。另外，同时新增多个字段时不能指定字段的位置。

下面在 my_goods 数据表中字段 name 后新增一个 num 字段，表示商品的数量，具体 SQL 语句如下。

```
mysql> ALTER TABLE my_goods ADD num INT AFTER name;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

执行上述 SQL 语句后，查看新增的字段，具体结果如下。

```
mysql> DESC my_goods;
+--------+----------------+-------+------+----------+--------+
| Field  | Type           | Null  | Key  | Default  | Extra  |
+--------+----------------+-------+------+----------+--------+
| id     | int(11)        | YES   |      | NULL     |        |
| name   | varchar(32)    | YES   |      | NULL     |        |
| num    | int(11)        | YES   |      | NULL     |        |
| des    | varchar(255)   | YES   |      | NULL     |        |
| price  | int(11)        | YES   |      | NULL     |        |
+--------+----------------+-------+------+----------+--------+
5 rows in set (0.00 sec)
```

5. 删除字段

删除字段指的是将某个字段从数据表中删除，MySQL 中可以通过 DROP 完成。基本语法格式如下。

```
ALTER TABLE 数据表名 DROP [COLUMN] 字段名;
```

下面以删除 my_goods 数据表中 num 字段为例演示，具体 SQL 语句如下。

```
mysql> ALTER TABLE my_goods DROP num;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

执行上述 SQL 语句后，查看删除 num 字段后数据表中的字段，具体结果如下。

```
mysql> DESC my_goods;
+--------+----------------+-------+------+----------+--------+
| Field  | Type           | Null  | Key  | Default  | Extra  |
+--------+----------------+-------+------+----------+--------+
|id      | int(11)        | YES   |      | NULL     |        |
|name    | varchar(32)    | YES   |      | NULL     |        |
|des     | varchar(255)   | YES   |      | NULL     |        |
|price   | int(11)        | YES   |      | NULL     |        |
+--------+----------------+-------+------+----------+--------+
4 rows in set (0.00 sec)
```

2.2.6　删除数据表

删除数据表操作指的是删除指定数据库中已经存在的表。另外，在删除数据表的同时，存储在数据表中的数据都将被删除，基本语法格式如下。

```
DROP [TEMPORARY] TABLE [IF EXISTS] 数据表 1 [, 数据表 2] …;
```

从上述语法可知，删除数据表时，可同时删除多个数据表，多个数据表之间使用逗号分隔。可选项 IF EXISTS 用于在删除一个不存在的数据表时，防止产生错误。

下面以删除数据表 my_goods 为例进行演示，具体 SQL 语句及执行结果如下。

```
mysql>DROP TABLE IF EXISTS my_goods;
Query OK, 0 rows affected (0.01 sec)
```

值得一提的是，在开发时应谨慎使用数据表删除操作，因为数据表一旦删除，表中的所有数据都将被清除。

2.3 数据操作

2.3.1 添加数据

通常情况下，要想操作数据表中的数据，首先要保证数据表中存在数据。MySQL 中使用 INSERT 语句向数据表中添加数据。根据操作的不同目的一般可以分为两种，一种是为所有字段添加数据，另一种是为部分字段添加数据。下面将对这两种操作进行详细讲解。

1. 为所有字段添加数据

在 MySQL 中，为所有字段插入记录时，可以省略字段名称，严格按照数据表结构(字段的位置)插入对应的值，基本语法格式如下。

```
INSERT [INTO] 数据表名 {VALUES | VALUE}(值 1[, 值 2] …);
```

从上述语法可知，关键字 INTO 是可选项，VALUES 和 VALUE 可以任选一种，通常情况下使用 VALUES。值列表“值 1 [，值 2] …”中多个值之间使用逗号分隔。

下面为 goods 表添加一条商品记录，编号为 1，商品名为 notebook，售价是 4998 元，描述信息为 High cost performance，具体 SQL 语句如下。

```
mysql>INSERT INTO goods
    ->VALUES (1, 'notebook', 4998, 'High cost performance');
Query OK, 1 row affected (0.00 sec)
```

在上述 SQL 语句中，插入的数据顺序与创建数据表时对应的字段顺序相同，分别表示商品编号、商品名、价格和商品描述。

脚下留心

在 MySQL 中，若创建的数据表未指定字符集，则数据表及表中的字段将使用默认的字符集 latin1。因此，若用户插入的数据中含有中文，则会出现错误提示。例如，向 goods 表中输入含有中文的数据，具体 SQL 语句及执行结果如下。

```
mysql>INSERT INTO goods
    ->VALUES(2, '笔记本', 9998, '续航时间超过 10 个小时');
ERROR 1366 (HY000): Incorrect string value: '\xB1\xCA\xBC\xC7\xB1\xBE' for column 'name' at row 1
```

为了解决以上中文插入的问题，通常在创建数据表时添加表选项，设置数据表的字符集，如下所示。

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS]表名
(字段名 字段类型 [字段属性]…) [DEFAULT] {CHARACTER SET|CHARSET} [=] utf8;
```

在上述语法中，CHARACTER SET 与 CHARSET 是同义词，设置字符集时选取其一即可。其中，utf8 字符集支持世界上大多数国家的字符，通常推荐使用此字符集。

另外，对于已经添加数据的数据表，则可以通过 ALTER TABLE…CHANGE/MODIFY 完成对表字段字符集的设置，在使用时需要注意它们语法的不同。下面以修改 goods 表中 name 和 description 字段的字符集为例进行演示。具体 SQL 语句及执行结果如下。

```
mysql>ALTER TABLE goods
    ->MODIFY name VARCHAR(32) CHARACTER SET utf8,
    ->MODIFY description VARCHAR(255) CHARACTER SET utf8;
Query OK, 1 row affected (0.02 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

上述 SQL 语句中，在同时修改多个字段时，使用逗号(,)分隔。修改完成后，可以再次向 goods 表中插入以上含有中文的数据，可以看到 Query OK 成功插入的提示。

2. 为部分字段添加数据

除了为数据表中所有字段添加数据外，还可以通过指定字段名的方式增加数据。其中指定的字段名可以是数据表中全部的字段，也可以是部分的字段。基本语法格式如下。

```
INSERT [INTO] 数据表名 (字段名 1 [, 字段名 2] …)
{VALUES | VALUE} (值 1[, 值 2] …);
```

在上述语法中，“(字段名 1 [,字段名 2] …)”字段列表中，多个字段名之间使用逗号分隔，且字段名的编写顺序可与表结构(字段位置)不同，只需保证值列表“(值 1[,值 2] …)”中的数据与其相对应即可。

下面将编号为 3 的 Mobile phone 插入到 goods 数据表中。具体 SQL 语句及执行结果如下。

```
mysql> INSERT INTO goods (id, name) VALUES (3, 'Mobile phone');
Query OK, 1 row affected (0.00 sec)
```

上述 SQL 语句中，字段的名称在使用时不需要使用引号包裹。另外，未添加数据的字段系统会自动为该字段添加默认值 NULL(空的)。

除此之外，MySQL 中还提供了另外一种使用 INSERT 语句为指定字段添加数据的方式。基本语法格式如下。

```
INSERT [INTO]数据表名
SET 字段名 1 =值 1 [, 字段名 2 =值 2] …;
```

在上述语法中，“字段名 1”“字段名 2”表示待添加数据的字段名称，“值 1”“值 2”表示添加的数据。若在 SET 关键字后，为表中多个字段添加数据，在每对“字段名＝值”之间使用逗号(,)分隔即可。

例如，使用“INSERT…SET…”语法实现上述示例的功能，具体 SQL 语句如下。

```
INSERT INTO goods SET id = 3, name = 'Mobile phone';
```

需要注意的是，最后一个字段赋值后不需要添加逗号。

3. 一次添加多行数据

在实际开发中，向一张数据表中同时插入多条记录时，重复地书写以上 INSERT 指令，操作不仅烦琐，又不便于阅读。因此，可以使用 MySQL 提供的另外一种插入数据的语法完成多数据插入。基本语法格式如下。

```
INSERT [INTO] 数据表名 [(字段列表)]
{VALUES | VALUE} (值列表) [, (值列表)] …;
```

在上述语法中，多个“值列表”之间使用逗号(,)分隔。其中，“字段列表”在省略时，插入的数据需严格按照数据表创建的顺序插入，否则“值列表”插入的数据仅需与字段列表中的字段相对应即可。

例如，将以上一次插入一条记录的操作修改成以下形式，完成一次添加多行数据。

```
mysql> INSERT INTO goods VALUES
    -> (1, 'notebook', 4998, 'High cost performance'),
    -> (2, '笔记本', 9998, '续航时间超过 10 个小时'),
    -> (3, 'Mobile phone', NULL, NULL);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

需要注意的是，在多数据插入时，若一条数据插入失败，则整个插入语句都会失败。

2.3.2 查询数据

数据的查询操作是 MySQL 中最常用，也是最重要的功能之一。下面介绍 3 种最基本的数据查询方式，其他更复杂的操作会在本书的其他章节中详细地讲解。

1. 查询表中全部数据

查询数据表中所有字段的数据，可以使用星号“ * ”通配符代替数据表中的所有字段名，基本语法格式如下。

```
SELECT * FROM 数据表名;
```

下面利用以上语法查看 goods 表中插入的全部数据。具体 SQL 语句如下。

```
mysql> SELECT * FROM goods;
+------+--------------+-------+----------------------+
| id   | name         | price | description          |
```

```
+-------+--------------+-------+-----------------------+
|   1   | notebook     |  4998 | High cost performance |
|   2   | 笔记本       |  9998 | 续航时间超过 10 个小时 |
|   3   | Mobile phone | NULL  | NULL                  |
+-------+--------------+-------+-----------------------+
3 rows in set (0.00 sec)
```

2. 查询表中部分字段

查询数据时，可在 SELECT 语句的字段列表中指定要查询的字段。基本语法格式如下。

```
SELECT {字段名 1, 字段名 2, 字段名 3, …} FROM 数据表名;
```

上述语法中，字段列表“字段名 1，字段名 2，字段名 3，…”中，若列出数据表中所有的字段名，则表示查询表中全部数据。

下面仅查看 goods 表中 id 和 name 字段，具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, name FROM goods;
+-------+--------------+
| id    | name         |
+-------+--------------+
|   1   | notebook     |
|   2   | 笔记本       |
|   3   | Mobile phone |
+-------+--------------+
3 rows in set (0.00 sec)
```

3. 简单条件查询数据

在查询数据时，若想要查询出符合条件的相关数据记录时，可以使用 WHERE 实现。基本语法格式如下。

```
SELECT * |{字段名 1, 字段名 2, 字段名 3, …}
FROM 数据表名 WHERE 字段名 = 值;
```

上述语法表示获取“字段名”等于指定“值”的数据记录，数据的内容中可以是表的部分字段或全部字段。

下面获取 goods 表中 id 等于 1 的全部商品信息。具体 SQL 语句及执行结果如下。

```
mysql> SELECT * FROM goods WHERE id = 1;
+-------+----------+-------+-----------------------+
| id    | name     | price | description           |
+-------+----------+-------+-----------------------+
|   1   | notebook |  4998 | High cost performance |
+-------+----------+-------+-----------------------+
1 rows in set (0.00 sec)
```

2.3.3 修改数据

修改数据是数据库中常见的操作，通常用于对表中的部分记录进行修改。例如，商品在做活动时，需要在原价的基础上打折，此时就需要对商品价格的数据进行修改。MySQL提供了UPDATE语句修改数据。基本语法格式如下。

```
UPDATE 数据表名
SET 字段名 1=值 1 [, 字段名 2=值 2, …]
[WHERE 条件表达式]
```

上述语法中，若实际使用时没有添加WHERE条件，那么表中所有对应的字段都会被修改成统一的值，因此读者在修改数据时，请谨慎操作。

下面将goods表中编号为2的商品价格由9998元修改为5899元。具体SQL语句及执行结果如下。

```
mysql>UPDATE goods SET price =5899 WHERE id =2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql>SELECT * FROM goods WHERE id =2;
+------+--------+-------+---------------------+
| id   | name   | price | description         |
+------+--------+-------+---------------------+
|   2  | 笔记本 |  5899 | 续航时间超过 10 个小时 |
+------+--------+-------+---------------------+
1 rows in set (0.00 sec)
```

执行完上述SQL语句后，使用SELECT查看编号为2的商品价格修改情况。

2.3.4 删除数据

删除数据是指对表中存在的记录进行删除。例如，商品停产后，可以删除商品表中的相关数据。MySQL中使用DELETE语句删除表中的记录，基本语法格式如下。

```
DELETE FROM 数据表名 [WHERE 条件表达式];
```

在上述语法中，“数据表名”指定要执行删除操作的表，WHERE条件为可选参数，用于设置删除的条件，满足条件的记录会被删除。

下面删除goods表中编号等于3的商品数据。具体SQL语句如下。

```
mysql>DELETE FROM goods WHERE id =3;
Query OK, 1 row affected (0.00 sec)
mysql>SELECT * FROM goods;
+------+----------+-------+-----------------------+
| id   | name     | price | description           |
+------+----------+-------+-----------------------+
|   1  | notebook |  4998 | High cost performance |
+------+----------+-------+-----------------------+
1 rows in set (0.00 sec)
```

执行上述 SQL 语句后，使用 SELECT 查询 goods 表中记录的变化。需要注意的是，在删除数据时若未指定 WHERE 条件，系统就会自动删除该表中所有的记录，因此读者在操作时需要慎重。

2.4　动手实践：电子杂志订阅表的操作

数据库的学习在于多看、多学、多想、多动手，只有将理论与实际相结合，才能够体现出数据开发与管理的重要性，展现知识学习的价值与力量。接下来请结合本章所学的知识完成电子杂志订阅表的操作。

【实践目标】

此实践的目标就是能够根据文字提示，完成对应数据表的创建，并可以对数据表中的数据进行简单的增、删、改、查操作。

【实践需求】

（1）在 mydb 数据库中创建一张电子杂志订阅表（subscribe）。

（2）电子杂志订阅表中要包含 4 个字段，分别为编号（id）、订阅邮件的邮箱地址（email）、用户是否确认订阅（status，使用数字表示，1 表示已确认，0 表示未确认）、邮箱确认的验证码（code）。

（3）为电子杂志订阅表添加 5 条测试数据，如表 2-2 所示。

表 2-2　测试数据信息

编号	邮箱地址	是否确认的状态	邮箱确认验证码
1	tom123@163.com	1	TRBXPO
2	lucy123@163.com	1	LOICPE
3	lily123@163.com	0	JIXDAMI
4	jimmy123@163.com	0	QKOLPH
5	joy123@163.com	1	JSMWNL

（4）查看已经通过邮箱确认的电子杂志订阅信息。

（5）将编号等于 4 的订阅确认状态设置为“已确认”。

（6）删除编号等于 5 的电子杂志订阅信息。

【动手实践】

1. 选择数据库

当 MySQL 服务器中不存在 mydb 数据库时，创建此数据库，否则忽略。在 mydb 数据库已经存在后，选择 mydb 数据库。具体 SQL 语句如下。

```
mysql> CREATE DATABASE IF NOT EXISTS mydb;
Query OK, 1 row affected, 1 warning (0.00 sec)
mysql> USE mydb;
Database changed
```

2. 创建电子杂志订阅表

根据【实践需求】第(2)条需求，创建电子杂志订阅表，具体 SQL 语句及执行结果如下。

```
mysql> CREATE TABLE subscribe (
    ->    id INT COMMENT '编号',
    ->    email VARCHAR(60) COMMENT '邮件订阅的邮箱地址',
    ->    status INT COMMENT '是否确认,0未确认,1已确认',
    ->    code VARCHAR(10) COMMENT '邮箱确认的验证码'
    -> ) DEFAULT CHARSET=utf8;
Query OK, 0 rows affected (0.01 sec)
```

3. 操作电子杂志订阅表的数据

(1) 添加数据。根据【实践需求】第(3)条给出的信息完成电子杂志订阅数据的新增，具体 SQL 语句及执行结果如下。

```
mysql> INSERT INTO subscribe VALUES
    -> (1, 'tom123@163.com', 1, 'TRBXPO'),
    -> (2, 'lucy123@163.com', 1, 'LOICPE'),
    -> (3, 'lily123@163.com', 0, 'JIXDAMI'),
    -> (4, 'jimmy123@163.com', 0, 'QKOLPH'),
    -> (5, 'joy123@163.com', 1, 'JSMWNL');
Query OK, 5 row affected (0.00 sec)
mysql> SELECT * FROM subscribe;
+------+------------------+--------+---------+
| id   | email            | status | code    |
+------+------------------+--------+---------+
|  1   | tom123@163.com   |   1    | TRBXPO  |
|  2   | lucy123@163.com  |   1    | LOICPE  |
|  3   | lily123@163.com  |   0    | JIXDAMI |
|  4   | jimmy123@163.com |   0    | QKOLPH  |
|  5   | joy123@163.com   |   1    | JSMWNL  |
+------+------------------+--------+---------+
5 rows in set (0.00 sec)
```

执行上述 SQL 语句后，使用 SELECT 查询表中的所有数据。

(2) 查询数据。查看已经通过邮箱确认的电子杂志订阅信息。具体 SQL 语句及执行结果如下。

```
mysql> SELECT * FROM subscribe WHERE status =1;
+------+-----------------+--------+--------+
| id   | email           | status | code   |
```

```
+-------+-----------------+--------+--------+
|   1   | tom123@163.com  |    1   | TRBXPO |
|   2   | lucy123@163.com |    1   | LOICPE |
|   5   | joy123@163.com  |    1   | JSMWNL |
+-------+-----------------+--------+--------+
3 rows in set (0.00 sec)
```

以上操作成功查询出了已经确认的邮箱。

（3）修改数据。将编号等于 4 的确认状态设置为已确认。具体 SQL 语句及执行结果如下。

```
mysql>UPDATE subscribe SET status =1 WHERE id =4;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql>SELECT * FROM subscribe WHERE id =4;
+-------+------------------+--------+--------+
| id    | email            | status | code   |
+-------+------------------+--------+--------+
|   4   | jimmy123@163.com |    1   | QKOLPH |
+-------+------------------+--------+--------+
1 row in set (0.00 sec)
```

执行上述 SQL 语句后，使用 SELECT 可查询出表中 id 等于 4 的记录 status 已修改为 1。

（4）删除数据。删除编号等于 5 的电子杂志订阅信息，具体 SQL 语句及执行结果如下。

```
mysql>DELETE FROM subscribe WHERE id =5;
Query OK, 1 row affected (0.00 sec)
mysql>SELECT * FROM subscribe;
+-------+------------------+--------+---------+
| id    | email            | status | code    |
+-------+------------------+--------+---------+
|   1   | tom123@163.com   |    1   | TRBXPO  |
|   2   | lucy123@163.com  |    1   | LOICPE  |
|   3   | lily123@163.com  |    0   | JIXDAMI |
|   4   | jimmy123@163.com |    1   | QKOLPH  |
+-------+------------------+--------+---------+
4 rows in set (0.00 sec)
```

从上述结果可以看出，SELECT 查询结果中，id 为 5 的记录不存在，说明已经被删除。

2.5　本章小结

本章主要讲解了数据库的创建、查看、选择与删除，数据表和数据的基础操作。通过本章的学习希望读者能够掌握 MySQL 的数据库、数据表以及数据的基本操作，为以后的学习和开发奠定夯实的基础。

2.6 课后练习

一、填空题

1. 添加________可在创建的数据库已存在时防止程序报错。
2. MySQL 提供的________可查看指定数据库的创建信息。
3. ________和________可在 MySQL 中添加注释内容，且在服务器运行时会被忽略。
4. 语句________可同时修改多个数据表名。
5. 查询数据时，通配符________可表示数据表中的所有字段。

二、判断题

1. 临时表仅在当前会话可见，会话关闭时会自动删除。(　　)
2. 仅修改数据表中的字段名称时，通常使用 ALTER TABLE…MODIFY 实现。(　　)
3. 修改数据时若未带 WHERE 条件，则表中对应字段都会被改为统一的值。(　　)
4. 数据库目录中的 db.opt 用于保存该数据库下的所有数据表信息。(　　)
5. 插入数据前必须使用 USE 选择操作的数据库。(　　)

三、选择题

1. 下列选项中，(　　)语句可查看数据表的创建时间。
   A. SHOW TABLES　　B. DESC 数据表名
   C. SHOW TABLE STATUS　　D. SHOW CREATE TABLE 数据表名
2. 若数据库中存在以下数据表，语句 SHOW TABLES LIKE 'sh_'的结果为(　　)。
   A. fish　　B. mydb　　C. she　　D. unshift
3. 下面语法不能实现新增数据的是(　　)。
   A. INSERT 表名 VALUE(值列表)
   B. INSERT INTO 表名 VALUE(值列表)
   C. INSERT INTO 表名 VALUES(值列表)
   D. INSERT INTO 表名(值列表)
4. 语句(　　)可以删除数据表中指定条件的数据。
   A. DELETE　　B. DROP
   C. ALTER TABLE　　D. 以上答案全部正确
5. 语句 ALTER TABLE…MODIFY 添加(　　)可将字段调整为数据表的第 1 个字段。
   A. FIRST 字段名　　B. FIRST
   C. AFTER 字段名　　D. AFTER

四、实训题

1. 按下列表结构，利用 SQL 语句在 mydb 数据库中创建 topic 表。

字　段　名	数 据 类 型	备　　注
id	INT	专题编号
title	VARCHAR(255)	专题名称
intro	VARCHAR(255)	专题介绍
start_time	INT	专题开始时间
end_time	INT	专题结束时间

2. 为 mydb.goods 表新增 total(库存量)和 add_time(发布时间)字段。

第3章 数据类型与约束

学习目标

- 掌握 MySQL 中常用数据类型的使用
- 掌握 MySQL 中常用约束的使用
- 掌握 MySQL 中字符集的设置与处理

在数据库中，数据表用来组织和保存各种数据，它是由表结构和数据组成的。在设计表结构时，经常需要根据实际需求，选择合适的数据类型和约束。本章将围绕数据类型和约束进行详细讲解。

3.1 数据类型

使用 MySQL 数据库存储数据时，不同的数据类型决定了 MySQL 存储数据方式的不同。MySQL 数据库提供了多种数据类型，其中包括数字类型、时间和日期类型、字符串类型。本节将针对这些数据类型进行讲解。

3.1.1 数字类型

在数据库中，经常需要存储一些数字，如商品的库存、销量、价格等，适合用数字类型来保存。数字类型包括整数类型、浮点数类型、定点数类型、BIT(位)类型等，下面分别进行讲解。

1. 整数类型

MySQL 中的整数类型用于保存整数。根据取值范围的不同，整数类型可分为 5 种，分别是 TINYINT、SMALLINT、MEDIUMINT、INT 和 BIGINT。不同整数类型所对应的字节大小和取值范围如表 3-1 所示。

表 3-1 MySQL 整数类型

数据类型	字节数	无符号数的取值范围	有符号数的取值范围
TINYINT	1	0～255	－128～127
SMALLINT	2	0～65 535	－32 768～32 767
MEDIUMINT	3	0～16 777 215	－8 388 608～8 388 607
INT	4	0～4 294 967 295	－2 147 483 648～2 147 483 647
BIGINT	8	0～18 446 744 073 709 551 615	－9 223 372 036 854 775 808～9 223 372 036 854 775 807

从表 3-1 中可以看出，不同整数类型所占用的字节数和取值范围都是不同的。其中，占用字节数最小的是 TINYINT，占用字节数最大的是 BIGINT。不同整数类型的取值范围可以根据字节数计算出来，例如，TINYINT 类型的整数占用 1 字节，1 字节是 8 位，那么，TINYINT 类型无符号数的最大值就是 2^8-1（即 255），有符号数的最大值就是 2^7-1（即 127）。同理，可以算出其他不同整数类型的取值范围。

需要注意的是，若使用无符号数据类型，需要在数据类型右边加上 UNSIGNED 关键字来修饰，例如，INT UNSIGNED 表示无符号 INT 类型。

为了让读者更好理解，下面通过案例的方式演示整数类型的使用及注意事项。

（1）创建 my_int 表，选取 INT 和 TINYINT 两种类型测试。具体 SQL 语句如下。

```
mysql>USE mydb;
mysql>CREATE TABLE my_int (
    ->    int_1 INT,
    ->    int_2 INT UNSIGNED,
    ->    int_3 TINYINT,
    ->    int_4 TINYINT UNSIGNED
    ->);
```

上述 SQL 语句中，int_1 和 int_3 是有符号类型，int_2 和 int_4 是无符号类型。

（2）插入记录进行测试。当数值在合法的取值范围内时，可以正确插入，反之则无法插入，提示错误信息。具体 SQL 语句及执行结果如下。

```
#① 插入成功测试
mysql>INSERT INTO my_int VALUES(1000, 1000, 100, 100);
Query OK, 1 row affected (0.00 sec)
#② 插入失败测试
mysql>INSERT INTO my_int VALUES(1000,-1000, 100, 100);
ERROR 1264 (22003): Out of range value for column 'int_2' at row 1
```

从上述结果可以看出，由于“-1000”超出了无符号 INT 类型的取值范围，数据插入失败，MySQL 显示了错误信息，提示 int_2 字段超出取值范围。

（3）查看 my_int 表的结构，具体 SQL 语句及执行结果如下。

```
mysql>DESC my_int;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| int_1 | int(11)              | YES  |     | NULL    |       |
| int_2 | int(10) unsigned     | YES  |     | NULL    |       |
| int_3 | tinyint(4)           | YES  |     | NULL    |       |
| int_4 | tinyint(3) unsigned  | YES  |     | NULL    |       |
+-------+----------------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

在执行结果中，数据类型右边使用小括号数字标注了显示宽度。默认情况下，显示宽度是取值范围所能表示的最大宽度。对于有符号类型，符号也占用一个宽度。例如，255 的显示宽度为 3，-128 的显示宽度为 4。需要注意的是，显示宽度与取值范围无关，若数值的位

数小于显示宽度，会填充空格，若大于显示宽度，则不影响显示结果。

(4) 为字段设置零填充(ZEROFILL)时，若数值宽度小于显示宽度，会在左侧填充 0。创建 my_int2 表，为字段设置零填充和宽度，具体 SQL 语句执行结果如下。

```
mysql> CREATE TABLE my_int2 (
    ->    int_1 INT(3) ZEROFILL,
    ->    int_2 TINYINT(6) ZEROFILL
    -> );
Query OK, 0 rows affected (0.01 sec)
mysql> DESC my_int2;
+-------+------------------------------+------+-----+---------+-------+
| Field | Type                         | Null | Key | Default | Extra |
+-------+------------------------------+------+-----+---------+-------+
| int_1 | int(3) unsigned zerofill     | YES  |     | NULL    |       |
| int_2 | tinyint(6) unsigned zerofill | YES  |     | NULL    |       |
+-------+------------------------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

在上述结果中，设置零填充后，字段自动设为无符号类型，这是因为负数不能使用零填充。

(5) 插入数据测试，具体 SQL 语句及执行结果如下。

```
mysql> INSERT INTO my_int2 VALUES(1234, 2);
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM my_int2;
+-------+--------+
| int_1 | int_2  |
+-------+--------+
|  1234 | 000002 |
+-------+--------+
1 row in set (0.00 sec)
```

从上述结果可知，当数值超过显示宽度时，不填充零；当数值未达到显示宽度时，在左侧填充零。

小提示：

(1) 在选择数据类型时，若一个数据将来可能参与数学计算，推荐使用整数、浮点数或定点数类型；若只用来显示，则推荐使用字符串类型。例如，商品库存可能需要增加、减少、求和等，所以保存为整数类型；用户的身份证、电话号码一般不需要计算，可以保存为字符串型。

(2) 表的主键推荐使用整数类型，与字符串相比，整数类型的处理效率更高，查询速度更快。

(3) 当插入的值的数据类型与字段的数据类型不一致，或使用 ALTER TABLE 修改字段的数据类型时，MySQL 会尝试尽可能将现有的值转换为新类型。例如，字符串'123'、'−123'、'1.23'与数字 123、−123、1.23 可以互相转换；1.5 转换为整数时，会被四舍五入，结果为 2。

2. 浮点数类型

在 MySQL 中，存储的小数都是使用浮点数或定点数来表示的。浮点数的类型有两种，分别是单精度浮点数类型（FLOAT）和双精度浮点数类型（DOUBLE），对应的字节大小及其取值范围如表 3-2 所示。

表 3-2　MySQL 浮点数类型

数据类型	字节数	负数的取值范围	非负数的取值范围
FLOAT	4	－3.402 823 466E＋38～－1.175 494 351E－38	0 和 1.175 494 351E－38～3.402 823 466E＋38
DOUBLE	8	－1.797 693 134 862 315 7E＋308～－2.225 073 858 507 201 4E－308	0 和 2.225 073 858 507 201 4E－308～1.797 693 134 862 315 7E＋308

表 3-2 中列举的取值范围是理论上的极限值，但根据不同的硬件或操作系统，实际范围可能会小。另外，当浮点数类型使用 UNSIGNED 修饰为无符号时，取值范围将不包含负数。

需要注意的是，浮点数类型虽然取值范围很大，但是精度并不高。FLOAT 的精度为 6 位或 7 位，DOUBLE 的精度大约为 15 位。如果超出精度，可能会导致给定的数值与实际保存的数值不一致，发生精度损失。

为了让读者更好地理解，下面通过案例的方式演示浮点数类型的使用及注意事项。具体 SQL 语句及执行结果如下。

```
#① 创建表,选取 FLOAT 类型进行测试
mysql> CREATE TABLE my_float (f1 FLOAT, f2 FLOAT);
Query OK, 0 rows affected (0.01 sec)
#② 插入未超出精度的数字
mysql> INSERT INTO my_float VALUES(111111, 1.11111);
Query OK, 1 row affected (0.00 sec)
#③ 插入超出精度的数字
mysql> INSERT INTO my_float VALUES(1111111, 1.111111);
Query OK, 1 row affected (0.00 sec)
#④ 插入 7 位数,第 7 位四舍五入
mysql> INSERT INTO my_float VALUES(1111114, 1111115);
Query OK, 1 row affected (0.00 sec)
#⑤ 插入 8 位数,第 7 位四舍五入,第 8 位忽略
mysql> INSERT INTO my_float VALUES(11111149, 11111159);
Query OK, 1 row affected (0.00 sec)
#⑥ 查询结果
mysql> SELECT * FROM my_float;
+----------+----------+
| f1       | f2       |
+----------+----------+
|   111111 | 1.11111  |
|  1111110 | 1.11111  |
|  1111110 | 1111120  |
```

```
| 11111100    | 11111200    |
+----------+---------+
4 rows in set (0.00 sec)
```

从上述结果可以看出，当一个数字的整数部分和小数部分加起来达到7位时，第7位就会四舍五入。

3. 定点数类型

定点数类型(DECIMAL)通过DECIMAL(M,D)设置位数和精度，其中，M表示数字总位数(不包括"."和"—")，最大值为65，默认值为10；D表示小数点后的位数，最大值为30，默认值为0。例如，DECIMAL(5,2)表示的取值范围是—999.99～999.99。系统会自动根据存储的数据来分配存储空间。若不允许保存负数，可通过UNSIGNED修饰。

为了让读者更好地理解，下面通过案例的方式演示定点数类型的使用及注意事项。具体SQL语句及执行结果如下。

```
#① 创建表,选取 DECIMAL 类型进行测试
mysql> CREATE TABLE my_decimal (d1 DECIMAL(4,2), d2 DECIMAL(4,2));
Query OK, 0 rows affected (0.01 sec)
#② 插入的小数部分超出范围时,会四舍五入并出现警告
mysql> INSERT INTO my_decimal VALUES(1.234, 1.235);
Query OK, 1 row affected, 2 warnings (0.00 sec)
mysql> SHOW WARNINGS;
+-------+------+----------------------------------------+
| Level   | Code   | Message                                  |
+-------+------+----------------------------------------+
| Note    | 1265   | Data truncated for column 'd1' at row 1  |
| Note    | 1265   | Data truncated for column 'd2' at row 1  |
+-------+------+----------------------------------------+
2 rows in set (0.00 sec)
#③ 插入的小数部分四舍五入导致整数部分进位时,插入失败
mysql> INSERT INTO my_decimal VALUES(99.99, 99.999);
ERROR 1264 (22003): Out of range value for column 'd2' at row 1
#④ 查询结果
mysql> SELECT * FROM my_decimal;
+------+------+
| d1     | d2     |
+------+------+
| 1.23   | 1.24   |
+------+------+
1 row in set (0.00 sec)
```

从上述结果可以看出，若小数部分超出范围，会进行四舍五入，并出现Data truncated(数据截断)警告；若整数部分超出范围，数据会插入失败，提示Out of range value(超出取值范围)错误。

小提示： 浮点数类型也可以设置位数和精度，如float(8,2)，但仍有可能损失精度。在实际使用时应避免使用浮点数类型，以免出现不能人为控制的问题。因此，对于小数类型的

设置，推荐使用定点数类型并设置合理的范围可以使计算更为准确。

4. BIT 类型

BIT(位)类型用于存储二进制数据，语法为 BIT(M)，M 表示位数，范围为 1～64。

下面以保存字符 A 为例，A 的 ASCII 码为十进制 65，对应的二进制为 1000001，总共有 7 位，需要至少 7 位来保存。示例 SQL 语句如下。

```
#① 获取字符"A"的 ASCII 码,结果为 65
mysql> SELECT ASCII('A');
#② 将十进制数转换为二进制,并计算长度,结果分别为 1000001、7
mysql> SELECT BIN(65), LENGTH(BIN(65));
#③ 创建表,然后插入数据
mysql> CREATE TABLE my_bit (b BIT(7));
mysql> INSERT INTO my_bit VALUES(65);
#④ 查询数据,查询结果为"A"
mysql> SELECT * FROM my_bit WHERE b = 65;
#⑤ 查询数据并转为二进制数字显示,查询结果为"1000001"
mysql> SELECT BIN(b) FROM my_bit;
```

从上述示例可以看出，利用 MySQL 中的 ASCII()、BIN()、LENGTH()函数可以方便地查询 ASCII 码、二进制值和数字长度。BIT 类型字段在数字插入时转换为二进制保存，但在利用 SELECT 查询时，会自动转换为对应的字符显示。

多学一招：MySQL 中的直接常量

直接常量是指在 MySQL 中直接编写的字面常量，如数字 123、字符串'abc'等，常用于在 INSERT 语句中编写插入的数据。直接常量有多种语法形式，具体如下。

(1) 十进制数：语法近似于日常生活中的数字，如 123、1.23、－1.23，以及科学计数法 1E2、1E－2(E 不分大小写)。

(2) 二进制数：在二进制字符串前加前缀 b，形如"b'1000001'"。通过"SELECT b'1000001';"可查看二进制转为 ASCII 字符后的结果，即字符 A。

(3) 十六进制数：有两种表示方式，形如"x'41'"和"0x41"。其中，十六进制数 41 对应十进制数为 65。通过"SELECT HEX(65);"可查看十进制 65 转为十六进制的结果，即 41；通过"SELECT x'41', 0x41;"可查看 ASCII 字符，即字符 A。

(4) 字符串：MySQL 支持单引号和双引号定界符，形如'abc'和"abc"，推荐使用单引号定界符。若要在单/双引号字符串中书写单/双引号，需要在单/双引号前面加上反斜线"\"转义，即"\'"和"\""，这种方式称为转义字符。常用的转义字符如表 3-3 所示。

表 3-3　常用转义字符

转义字符	含义	转义字符	含义
\0	空字符(NUL)	\t	制表符(HT)
\r	回车符(CR)	\b	退格(BS)
\n	换行符(LF)	\'	单引号

续表

转义字符	含义	转义字符	含义
\"	双引号	\%	%(常用于 LIKE 条件)
\\	反斜线	_	_(常用于 LIKE 条件)

(5) 布尔值：有 TRUE 和 FALSE 两个值(不分大小写)，通常用于逻辑判断，表示事物的“真”和“假”。在 SELECT、INSERT 等语句中使用布尔值时，TRUE 会转换为 1，FALSE 会转换为 0。

(6) NULL 值：通常用来表示没有值、值不确定等含义。例如，在插入一条商品数据时，暂时不知道该商品的库存量，可将库存量设为 NULL，以后再修改。

3.1.2 时间和日期类型

为了方便在数据库中存储日期和时间，MySQL 提供了表示日期和时间的数据类型，分别是 YEAR、DATE、TIME、DATETIME 和 TIMESTAMP。表 3-4 列举了这些 MySQL 中日期和时间数据类型所对应的字节数、取值范围、日期格式以及零值。

表 3-4 MySQL 日期和时间类型

数据类型	取值范围	日期格式	零 值
YEAR	1901～2155	YYYY	0000
DATE	1000-01-01～9999-12-3	YYYY-MM-DD	0000-00-00
TIME	-838:59:59～838:59:59	HH:MM:SS	00:00:00
DATETIME	1000-01-01 00:00:00～ 9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	0000-00-00 00:00:00
TIMESTAMP	1970-01-01 00:00:01～ 2038-01-19 03:14:07	YYYY-MM-DD HH:MM:SS	0000-00-00 00:00:00

在表 3-4 中，日期格式 YYYY 表示年，MM 表示月，DD 表示日。每种日期和时间类型的取值范围都是不同的。需要注意的是，如果插入的数值不合法，系统会自动将对应的零值插入数据库中。

为了让读者更好地学习日期和时间类型，接下来分别进行详细讲解。

1. YEAR 类型

YEAR 类型用于表示年份，使用示例如下。

```
CREATE TABLE my_year (y YEAR);              #设置 y 字段的数据类型为 YEAR
INSERT INTO my_year VALUES (2020);          #插入年份数据，2020 年
```

在 MySQL 中，可以使用以下 3 种格式指定 YEAR 类型的值。

(1) 使用 4 位字符串或数字表示，为'1901'～'2155'或 1901～2155。例如，输入'2020'或 2020，插入到数据库中的值均为 2020。

(2) 使用两位字符串表示为'00'～'99'，其中，'00'～'69'的值会被转换为 2000～2069 的

YEAR 值，'70'～'99'的值会被转换为 1970～1999 的 YEAR 值。例如，输入'20'，插入到数据库中的值为 2020。

(3) 使用两位数字表示为 1～99，其中，1～69 的值会被转换为 2001～2069 的 YEAR 值，70～99 的值会被转换为 1970～1999 的 YEAR 值。例如，输入 20，插入到数据库中的值为 2020。

需要注意的是，当使用 YEAR 类型时，一定要区分'0'和 0。因为字符串格式的'0'表示的 YEAR 值是 2000，而数字格式的 0 表示的 YEAR 值是 0000。

2. DATE 类型

DATE 类型用于表示日期值，不包含时间部分，使用示例如下。

```
CREATE TABLE my_date (d DATE);                      #设置 d 字段的数据类型为 DATE
INSERT INTO my_date VALUES ('2020-01-21');          #插入日期数据
INSERT INTO my_date VALUES (CURRENT_DATE);          #插入当前系统日期
INSERT INTO my_date VALUES (NOW());                 #插入当前系统日期
```

在 MySQL 中，可以使用以下 4 种格式指定 DATE 类型的值。

(1) 以'YYYY-MM-DD'或者'YYYYMMDD'字符串格式表示。

例如，输入'2020-01-21'或'20200121'，插入数据库中的日期都为 2020-01-21。

(2) 以'YY-MM-DD'或者'YYMMDD'字符串格式表示。YY 表示的是年，为'00'～'99'，其中'00'～'69'的值会被转换为 2000～2069 的值，'70'～'99'的值会被转换为 1970～1999 的值。

例如，输入'20-01-21'或'200121'，插入数据库中的日期都为 2020-01-21。

(3) 以 YY-MM-DD 或者 YYMMDD 数字格式表示。

例如，输入 20-01-21 或 200121，插入数据库中的日期都为 2020-01-21。

(4) 使用 CURRENT_DATE 或者 NOW()输入当前系统日期。

小提示：

(1) 通过"SELECT CURRENT_DATE;"或"SELECT NOW();"可查看当前日期。

(2) 日期中的分隔符"-"还可以用"."","/"等符号替代。

3. TIME 类型

TIME 类型用于表示时间值，它的显示形式一般为 HH:MM:SS，其中，HH 表示小时，MM 表示分，SS 表示秒。在 MySQL 中，可以使用以下 3 种格式指定 TIME 类型的值。

(1) 以'HHMMSS'字符串或者 HHMMSS 数字格式表示。

例如，输入'345454'或 345454，插入数据库中的时间为 34:54:54(34 小时 54 分 54 秒)。

(2) 以'D HH:MM:SS'字符串格式表示。其中，D 表示日，可以取 0～34 之间的值，插入数据时，小时的值等于(D×24＋HH)。

例如，输入'2 11:30:50'，插入数据库中的时间为 59:30:50；输入'11:30:50'，插入数据库中的时间为 11:30:50；输入'34 22:59:59'，插入数据库中的时间为 838:59:59。

(3) 使用 CURRENT_TIME 或 NOW()输入当前系统时间。

4. DATETIME 类型

DATETIME 类型用于表示日期和时间，它的显示形式为'YYYY-MM-DD HH:MM:SS'，其中，YYYY 表示年，MM 表示月，DD 表示日，HH 表示小时，MM 表示分，SS 表示秒。在 MySQL 中，可以使用以下 4 种格式指定 DATETIME 类型的值。

(1) 以'YYYY-MM-DD HH:MM:SS'或者'YYYYMMDDHHMMSS'字符串格式表示的日期和时间，取值范围为'1000-01-01 00:00:00'～'9999-12-31 23:59:59'。

例如，输入'2014-01-22 09:01:23'或 20140122090123，插入数据库中的 DATETIME 值都为 2014-01-22 09:01:23。

(2) 以'YY-MM-DD HH:MM:SS'或者'YYMMDDHHMMSS'字符串格式表示的日期和时间，其中 YY 表示年，取值范围为'00'～'99'。与 DATE 类型中的 YY 相同，'00'～'69'范围的值会被转换为 2000～2069 范围的值，'70'～'99'范围的值会被转换为 1970～1999 范围的值。

(3) 以 YYYYMMDDHHMMSS 或者 YYMMDDHHMMSS 数字格式表示的日期和时间。

例如，插入 20140122090123 或者 140122090123，插入数据库中的 DATETIME 值都为 2014-01-22 09:01:23。

(4) 使用 NOW()来输入当前系统的日期和时间。

5. TIMESTAMP 类型

TIMESTAMP(时间戳)类型用于表示日期和时间，它的显示形式与 DATETIME 相同，但取值范围比 DATETIME 小。下面介绍几种 TIMESTAMP 类型与 DATATIME 类型不同的形式，具体如下。

(1) 使用 CURRENT_TIMESTAMP 来输入系统当前日期和时间。

(2) 无任何输入，或输入 NULL 时，实际保存的是系统当前日期和时间。

脚下留心

在 MySQL 中，TIMESTAMP 字段默认情况下会自动设置 NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP 属性，具体解释如下。

(1) NOT NULL 表示非空约束，该字段将不允许保存 NULL 值。

(2) DEFAULT 表示默认约束，当字段无任何输入时，自动设置某个值作为默认值。此处设为 CURRENT_TIMESTAMP 表示使用系统当前日期和时间作为默认值。

(3) ON UPDATE 用于当一条记录中的其他字段被 UPDATE 语句修改时，自动更改该字段为某个值。此处设为 CURRENT_TIMESTAMP 表示每次修改时保存修改时的系统日期和时间。

若为 TIMESTAMP 字段手动设置 DEFAULT 属性时，该字段将不会自动设置 ON UPDATE 属性，通过如下 SQL 语句可进行测试。

```
CREATE TABLE my_timestamp (
  t1 TIMESTAMP,
  t2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  t3 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

上述 SQL 语句通过 t1、t2、t3 字段演示了 TIMESTAMP 字段的 3 种使用方式。其中，t1 和 t2 的设置结果是相同的，t3 没有设置 ON UPDATE 属性。

通过 DESC 查看表结构，会发现 t1 和 t2 字段具有 DEFAULT 和 ON UPDATE 属性，t3 字段没有 ON UPDATE 属性，如下所示。

```
mysql> DESC my_timestamp;
+-------+-----------+------+---+-----------------+----------------------------+
| Field | Type      | Null |Key| Default         | Extra                      |
+-------+-----------+------+---+-----------------+----------------------------+
| t1    | timestamp | NO   |   | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
| t2    | timestamp | NO   |   | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
| t3    | timestamp | NO   |   | CURRENT_TIMESTAMP |                            |
+-------+-----------+------+---+-----------------+----------------------------+
3 rows in set (0.00 sec)
```

值得一提的是，若使用 MySQL 5.6 之前的版本，my_timestamp 表会创建失败。这是因为 MySQL 5.6 之前的版本在一张表中只允许一个字段使用 CURRENT_TIMESTAMP 作为 DEFAULT 和 ON UPDATE 的值。此时可以分成多张表进行测试。

3.1.3　字符串类型

MySQL 中的字符串类型分为 CHAR、VARCHAR、TEXT 等多种类型，不同数据类型具有不同的特点，具体如表 3-5 所示。

表 3-5　MySQL 字符串类型

数据类型	类型说明
CHAR	固定长度字符串
VARCHAR	可变长度字符串
TEXT	大文本数据
ENUM	枚举类型
SET	字符串对象
BINARY	固定长度的二进制数据
VARBINARY	可变长度的二进制数据
BLOB	二进制大对象(Binary Large Object)

接下来，针对这些字符串类型进行详细讲解，具体如下。

1. CHAR 和 VARCHAR 类型

CHAR 和 VARCHAR 类型都用来保存字符串数据。不同的是，VARCHAR 可以存储可变长度的字符串。在 MySQL 中，定义 CHAR 和 VARCHAR 类型的方式如下所示。

```
CHAR(M)
```

或

```
VARCHAR(M)
```

在上述定义方式中，M 指的是字符串的最大长度。为了对比 CHAR 和 VARCHAR 之间的区别，下面以 CHAR(4)和 VARCHAR(4)为例进行说明，具体如表 3-6 所示。

表 3-6　CHAR(4)和 VARCHAR(4)对比

插　入　值	CHAR(4)存储需求	VARCHAR(4)存储需求
''	4 字节	1 字节
'ab'	4 字节	3 字节
'abc'	4 字节	4 字节
'abcd'	4 字节	5 字节

从表 3-6 中可以看出，对于 CHAR(4)无论插入值的长度是多少，所占用的存储空间都是 4 字节。而 VARCHAR(4)占用的字节数为实际长度加 1。

2. TEXT 类型

TEXT 类型用于保存大文本数据，例如，文章内容、评论等比较长的文本。它的类型分为 4 种，具体如表 3-7 所示。

表 3-7　TEXT 类型

数 据 类 型	存 储 范 围	数 据 类 型	存 储 范 围
TINYTEXT	$0\sim2^{8}-1$ 字节	MEDIUMTEXT	$0\sim2^{24}-1$ 字节
TEXT	$0\sim2^{16}-1$ 字节	LONGTEXT	$0\sim2^{32}-1$ 字节

TEXT 类型所能保存的最大字符数量取决于字符串实际占用的字节数。

脚下留心

(1) CHAR 和 VARCHAR 类型在插入数据时，若字符串末尾有空格，CHAR 类型会自动去掉空格后保存，而 VARCHAR、TEXT 类型会保留空格。

(2) 在使用"="等运算符对 CHAR、VARCHAR、TEXT 进行比较时，字符串末尾的空格会被忽略。例如，使用 WHERE 查询'a'字符串，查询结果中可能包含 a 后面有空格的情况，反之，若查询条件字符串末尾有空格(如 'a ')，空格也会被忽略。

(3) 由于默认情况下创建的数据库和表使用的校对集(latin1_swedish_ci)对大小写不敏感，因此 CHAR、VARCHAR、TEXT、ENUM、SET 类型都不区分大小写。例如，使用 WHERE 查询'a'字符串，则"a"和"A"都会被查询出来。而 BINARY、VARBINARY、BLOB

类型区分大小写，这是因为它们使用二进制方式保存数据。

(4) MySQL 默认规定一条记录的最大长度是 65535 字节，一般来说，字段分配的存储空间和额外开销加在一起不能超过 65535 字节，如果超过了这个限制，在 SQL 严格模式下表会创建失败，提示 Row size too large。但 TEXT 和 BLOB 类型字段的存储空间不受此限制，它们只占用额外开销(大约 12 字节)。

(5) 在没有超过 65535 限制的情况下，CHAR 字段的 M 最大值为 255，VARCHAR 字段的 M 的最大值取决于字符集，常用的字符集有 latin1(默认)、gbk 和 utf8，对应的 M 最大值分别为 65533、32766 和 21844，若表中只有一个字段且设置了非空约束，M 可达到最大值，否则 M 的最大值会减小。

(6) 从执行效率上来说，TEXT 和 BLOB 不如 CHAR、VARCHAR 类型，建议只有当需要保存大量数据时，才选择使用 TEXT 或 BLOB 类型。

多学一招：二进制方式比较字符串

若要在 CHAR、VARCHAR、TEXT 类型的字符串比较时严格区分大小写，有多种实现方式，接下来讲解两种比较常见的方式。

(1) 使用 BINARY 关键字。在字段名或某个值的前面加上 BINARY 关键字可以将类型转换为二进制，转换后进行比较就可以严格区分大小写和空格。具体示例如下。

```
#① 直接测试比较结果
SELECT 'a ' ='A';                                        #比较结果：1(相等)
SELECT BINARY 'a' ='A', 'a' =BINARY 'A';                 #比较结果：0(不相等)
SELECT BINARY 'A ' ='A';                                 #比较结果：0(不相等)
SELECT BINARY 'A' ='A', 'A' =BINARY 'A';                 #比较结果：1(相等)
#② 在查询条件中进行二进制比较
CREATE TABLE my_char (c CHAR(2));
INSERT INTO my_char VALUES('A');
SELECT c FROM my_char WHERE c ='a ';                     #查询结果为"A"
SELECT c FROM my_char WHERE BINARY c ='a';               #查询结果为空
SELECT c FROM my_char WHERE BINARY c ='A ';              #查询结果为空
SELECT c FROM my_char WHERE BINARY c ='A';               #查询结果为"A"
```

(2) 设置字段的校对集。latin1、gbk、utf8 编码默认的校对集分别为 latin1_swedish_ci、gbk_chinese_ci、utf8_general_ci，将其分别改为 latin1_bin、gbk_bin、utf8_bin 即可区分大小写，但在比较时仍会忽略字符串末尾的空格。具体示例如下。

```
#① 创建表时设置字段的校对集
CREATE TABLE my_char (
  c1 CHAR(2) CHARACTER SET latin1 COLLATE latin1_bin,
  c2 CHAR(2) CHARACTER SET gbk COLLATE gbk_bin,
  c3 CHAR(2) CHARACTER SET utf8 COLLATE utf8_bin
);
#② 插入测试数据
INSERT INTO my_char VALUES('A', 'A', 'A');
#③ 查询测试
SELECT c1 ='a', c2 ='a', c3 ='a' FROM my_char;           #结果：0(不相等)
SELECT c1 ='A', c2 ='A', c3 ='A' FROM my_char;           #结果：1(相等)
SELECT c1 ='A ',c2 ='A ',c3 ='A ' FROM my_char;          #结果：1(相等)
```

3. ENUM 类型

ENUM 类型又称为枚举类型，定义 ENUM 类型的方式如下所示。

```
ENUM('值 1', '值 2', '值 3', …, '值 n')
```

在上述格式中，('值 1', '值 2', '值 3',…, '值 n')称为枚举列表，ENUM 类型的数据只能从枚举列表中取，并且只能取一个。ENMU 类型的使用示例如下。

```
#① 创建表
mysql> CREATE TABLE my_enum (gender ENUM('male', 'female'));
#② 插入两条测试记录
mysql> INSERT INTO my_enum VALUES('male'), ('female');
#③ 查询记录，查询结果为"female"
mysql> SELECT * FROM my_enum WHERE gender = 'female';
#④ 插入枚举列表中没有的值测试
mysql> INSERT INTO my_enum VALUES('m');
ERROR 1265 (01000): Data truncated for column 'gender' at row 1
```

在 MySQL 中，枚举列表最多可以有 65535 个值，每个值都有一个顺序编号，实际保存在记录中的是顺序编号，而不是列表中的值，因此不必担心过长的值占用空间。但在使用 SELECT、INSERT 等语句进行操作时，仍然使用列表中的值。

4. SET 类型

SET 类型用于保存字符串对象，其定义格式与 ENUM 类型类似，具体如下。

```
SET('值 1', '值 2', '值 3', …, '值 n')
```

SET 类型的列表中最多可以有 64 个值，且列表中的每个值都有一个顺序编号，为了节省空间，实际保存在记录中的也是顺序编号，但在使用 SELECT、INSERT 等语句进行操作时，仍然要使用列表中的值。

SET 类型与 ENUM 的区别在于，它可以从列表中选择一个或多个值来保存，多个值之间用逗号"，"分隔。具体使用示例如下。

```
#① 创建表
mysql> CREATE TABLE my_set (hobby SET('book', 'game', 'code'));
#② 插入 3 条测试记录
mysql> INSERT INTO my_set VALUES(''), ('book'), ('book,code');
#③ 查询记录，查询结果为"book,code"
mysql> SELECT * FROM my_set WHERE hobby = 'book,code';
```

小提示：

(1) ENUM 类型类似于单选框，SET 类型类似于复选框。

(2) ENUM 和 SET 类型的优势在于规范数据本身，限定只能插入规定的数据项，节省了存储空间，查询速度比 CHAR、VARCHAR 类型快。

(3) ENUM 和 SET 类型列表中的值都可以使用中文，但必须设置支持中文的字符集，

例如 CREATE TABLE my_enum (gender ENUM('男', '女')) CHARSET=GBK;。

(4) ENUM 和 SET 类型在填写列表、插入值、查找值等操作时,都会自动忽略末尾的空格。

5. BINARY 和 VARBINARY 类型

BINARY 和 VARBINARY 类型类似于 CHAR 和 VARCHAR,不同的是,它们所表示的是二进制数据。定义 BINARY 和 VARBINARY 类型的方式如下所示。

```
BINARY(M)
```

或

```
VARBINARY(M)
```

在上述格式中,M 是指二进制数据的最大字节长度。BINARY 类型的长度是固定的,如果数据的长度不足最大长度,将在数据的后面用"\0"补齐,最终达到指定长度。例如,指定数据类型为 BINARY(3),当插入 a 时,实际存储的数据为"a\0\0",当插入 ab 时,实际存储的数据为"ab\0"。

接下来演示 BINARY 和 VARBINARY 的使用示例,具体如下所示。

```
#① 创建表,插入测试记录
mysql>CREATE TABLE my_binary (b1 BINARY(4), b2 VARBINARY(4));
mysql>INSERT INTO my_binary VALUES('abc', 'xyz');
#② 查询记录,两次查询结果分别为"abc\0"和"xyz"(\0 显示为空格)
mysql>SELECT b1 FROM my_binary WHERE b1 ='abc\0';
mysql>SELECT b2 FROM my_binary WHERE b2 ='xyz';
#③ 查询记录,由于区分大小写,查询结果都为空
mysql>SELECT b1 FROM my_binary WHERE b1 ='ABC\0';
mysql>SELECT b2 FROM my_binary WHERE b2 ='XYZ';
```

从上述示例可以看出,在查询 BINARY 类型时,查询条件字符串也需要加上"\0"填充符,否则查询不到该记录,并且 BINARY 和 VARBINARY 都区分大小写。

6. BLOB 类型

BLOB 类型用于保存数据量很大的二进制数据,如图片、PDF 文档等。BLOB 类型分为 4 种,具体如表 3-8 所示。

表 3-8　BLOB 类型

数据类型	存储范围	数据类型	存储范围
TINYBLOB	$0\sim2^{8}-1$ 字节	MEDIUMBLOB	$0\sim2^{24}-1$ 字节
BLOB	$0\sim2^{16}-1$ 字节	LONGBLOB	$0\sim2^{32}-1$ 字节

需要注意的是,BLOB 类型与 TEXT 类型很相似,但 BLOB 类型数据是根据二进制编码进行比较和排序,而 TEXT 类型数据是根据文本模式进行比较和排序。

接下来演示 BLOB 的使用示例,具体如下所示。

```
#① 创建表,插入测试记录
mysql>CREATE TABLE my_blob (b BLOB);
```

```
mysql> INSERT INTO my_blob VALUES ('data');
#② 查询记录,查询结果为"data"
mysql> SELECT b FROM my_blob WHERE b = 'data';
#③ 查询记录,由于区分大小写,查询结果为空
mysql> SELECT b FROM my_blob WHERE b = 'Data';
```

从上述示例可以看出,BLOB 类型在查询时区分大小写。

多学一招:JSON 数据类型

MySQL 从 5.7.8 版本开始提供了 JSON 数据类型。JSON 是一种轻量级的数据交换格式,由 JavaScript 语言发展而来,其本质是一个字符串。MySQL 中 JSON 类型值常见的表现方式有两种,分别为 JSON 数组和 JSON 对象,示例如下。

```
#① JSON 数组
["abc", 10, null, true, false]
#② JSON 对象
{"k1": "value", "k2": 10}
```

从上述示例可知,JSON 数组中保存的数据可以是任意类型。其中,JSON 数组使用"["和"]"符号实现,多个值之间使用逗号分隔,如 10 和 null;JSON 对象使用"{"和"}"符号实现,保存的数据是一组键值对,如 k1 和 k2 是键名(或称为属性名),而 value 和 10 是键名对应的值。

与直接使用 MySQL 字符串类型相比,JSON 数据类型具有自动验证格式、优化存储格式的优点。JSON 数据类型所需的空间大致与 LONGBLOB 或 LONGTEXT 相同,且不能有默认值。下面演示 JSON 数据类型的使用示例,具体如下所示。

```
#① 创建表,插入测试记录
mysql> CREATE TABLE my_json (j1 JSON, j2 JSON);
mysql> INSERT INTO my_json VALUES
    -> ('{"k1": "value", "k2": 10}', '["run", "sing"]');
#② 查询记录
mysql> SELECT * FROM my_json;
+---------------------------+------------------+
| j1                        | j2               |
+---------------------------+------------------+
| {"k1": "value", "k2": 10} | ["run", "sing"]  |
+---------------------------+------------------+
1 row in set (0.00 sec)
```

从上述示例可以看出,JSON 数据类型的字段以字符串的方式插入数据即可。

3.2 表的约束

为了防止数据表中插入错误的数据,MySQL 定义了一些维护数据库完整性的规则,即表的约束。常见约束分为 5 种,分别是默认约束、非空约束、主键约束、唯一约束和外键约束。外键约束比较复杂,涉及多表操作,将在后面的章节中讲解,本节主要讲解其余 4 种约

束的使用方法。

3.2.1 默认约束

默认约束用于为数据表中的字段指定默认值，即当在表中插入一条新记录时，如果没有给这个字段赋值，那么，数据库系统会自动为这个字段插入默认值。默认值是通过 DEFAULT 关键字定义的，其基本语法格式如下。

```
字段名 数据类型  DEFAULT 默认值;
```

需要注意的是，BLOB、TEXT 数据类型不支持默认约束。下面通过案例演示默认约束的使用及注意事项。

(1) 创建 my_default 表，准备 name 和 age 两个字段进行测试，为 age 添加默认约束，设置默认值为 18。

```
mysql> CREATE TABLE my_default (
    ->    name VARCHAR(10),
    ->    age INT UNSIGNED DEFAULT 18
    -> );
```

(2) 使用 DESC 查看表结构，结果如下所示。

```
mysql> DESC my_default;
+--------+-------------------+-------+------+----------+--------+
| Field  | Type              | Null  | Key  | Default  | Extra  |
+--------+-------------------+-------+------+----------+--------+
| name   | varchar(10)       | YES   |      | NULL     |        |
| age    | int(10) unsigned  | YES   |      | 18       |        |
+--------+-------------------+-------+------+----------+--------+
2 rows in set (0.00 sec)
```

(3) 插入记录进行测试，具体 SQL 语句及执行结果如下。

```
#① 在插入记录时省略 name 和 age 字段
mysql> INSERT INTO my_default VALUES();
Query OK, 1 row affected (0.00 sec)
#② 在插入记录时省略 age 字段
mysql> INSERT INTO my_default (name) VALUES('a');
Query OK, 1 row affected (0.00 sec)
#③ 在 age 字段中插入 NULL 值
mysql> INSERT INTO my_default VALUES('b', NULL);
Query OK, 1 row affected (0.00 sec)
#④ 在 age 字段中使用默认值
mysql> INSERT INTO my_default VALUES('c', DEFAULT);
Query OK, 1 row affected (0.00 sec)
#⑤ 查询结果
mysql> SELECT * FROM my_default;
+------+------+
| name | age  |
```

```
+------+------+
| NULL   |  18   |
| a      |  18   |
| b      | NULL  |
| c      |  18   |
+------+------+
4 rows in set (0.00 sec)
```

在上述示例中，由于 name 和 age 字段没有设置非空约束，在插入记录时省略了这两个字段的值，则分别使用默认值 NULL 和 18。为 age 字段设置默认值 18 后，插入 NULL 值，则保存结果为 NULL，不使用默认值。在为有默认值的字段指定数据时，可以通过 DEFAULT 关键字直接指定其使用默认值。

（4）为现有的表添加或删除默认约束，具体 SQL 语句及执行结果如下。

```
#① 删除默认约束
mysql>ALTER TABLE my_default MODIFY age INT UNSIGNED;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
#② 添加默认约束
mysql>ALTER TABLE my_default MODIFY age INT UNSIGNED DEFAULT 18;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

通过上述示例可以看出，使用 ALTER TABLE 修改列属性即可添加或删除默认约束。

3.2.2 非空约束

非空约束指的是字段的值不能为 NULL，在 MySQL 中，非空约束是通过 NOT NULL 定义的，其基本语法格式如下。

```
字段名 数据类型 NOT NULL;
```

为了让读者更好地理解，下面通过案例演示非空约束的使用及注意事项。

（1）创建 my_not_null 表，准备 n1、n2 和 n3 字段进行测试，为 n2 和 n3 设置非空约束，为 n3 设置默认值为 18。

```
mysql>CREATE TABLE my_not_null (
    ->    n1 INT,
    ->    n2 INT NOT NULL,
    ->    n3 INT NOT NULL DEFAULT 18
    ->);
```

（2）使用 DESC 查看表结构，结果如下所示。

```
mysql>DESC my_not_null;
+-------+---------+------+-----+---------+-------+
| Field  | Type      | Null  | Key  | Default    | Extra   |
```

```
+--------+----------+------+-----+----------+--------+
| n1     | int(11)  |YES   |     | NULL     |        |
| n2     | int(11)  |NO    |     | NULL     |        |
| n3     | int(11)  |NO    |     | 18       |        |
+--------+----------+------+-----+----------+--------+
3 rows in set (0.00 sec)
```

在上述结果中,Null 列的值为 NO 表示该字段添加了非空约束。需要注意的是,添加了非空约束的 n2 字段的 Default 为 NULL,表示未给该字段设置默认值,而不能将其理解为默认值为 NULL,否则在插入数据时,若 n2 字段为 NULL,MySQL 会报“Column 'n2' cannot be null”错误提示。另外,在创建数据表时,非空约束与值为 NULL 的默认约束(DEFULT NULL)不能同时存在,否则数据表在创建时会失败,提示“Invalid default value for 'n2'”错误。

(3) 插入记录进行测试,具体 SQL 语句及执行结果如下。

```
#① 省略 n2 字段,插入失败,提示 n2 没有默认值
mysql> INSERT INTO my_not_null VALUES();
ERROR 1364 (HY000): Field 'n2' doesn't have a default value
#② 将 n2 字段设为 NULL,插入失败,提示 n2 字段不能为 NULL
mysql> INSERT INTO my_not_null VALUES(NULL, NULL, NULL);
ERROR 1048 (23000): Column 'n2' cannot be null
#③ 将 n3 字段设为 NULL,插入失败,提示 n3 字段不能为 NULL
mysql> INSERT INTO my_not_null VALUES(NULL, 20, NULL);
ERROR 1048 (23000): Column 'n3' cannot be null
#④ 省略 n1 和 n3 字段,插入成功
mysql> INSERT INTO my_not_null (n2) VALUES(20);
Query OK, 1 row affected (0.00 sec)
#⑤ 查询结果
mysql> SELECT * FROM my_not_null;
+-------+----+----+
| n1    | n2 | n3 |
+-------+----+----+
| NULL  | 20 | 18 |
+-------+----+----+
1 row in set (0.00 sec)
```

在上述示例中,由于 n2 字段不能为 NULL 且没有默认值,在插入时不能插入 NULL 或省略该字段;n3 字段设置了默认值,在插入时可以省略该字段,但不能插入 NULL。

小提示: 为现有的表添加或删除非空约束的方式与默认约束类似,使用 ALTER TABLE 修改列属性即可。但若目标列中已经保存了 NULL 值,添加非空约束会失败,提示“Invalid use of NULL value”,只要将 NULL 值改为其他值即可解决。

3.2.3 唯一约束

唯一约束用于保证数据表中字段的唯一性,即表中字段的值不能重复出现。唯一约束是通过 UNIQUE 定义的,其基本语法格式如下所示。

```
#列级约束
字段名 数据类型 UNIQUE;
#表级约束
UNIQUE(字段名1,字段名2,…);
```

在上述语法格式中，列级约束和表级约束是MySQL中的两种定义约束的方式。列级约束定义在一个列上，只对该列起约束作用；表级约束是独立于列的定义，可以应用在一个表的多个列上。

为了让读者更好地理解，下面通过案例演示唯一约束的使用及注意事项。

(1) 创建my_unique_1表和my_unique_2表，分别通过列级约束和表级约束的方式添加唯一约束。具体SQL语句和执行结果如下。

```
#列级约束
mysql> CREATE TABLE my_unique_1 (
    ->   id INT UNSIGNED UNIQUE,
    ->   username VARCHAR(10) UNIQUE
    -> );
Query OK, 0 rows affected (0.01 sec)
#表级约束
mysql> CREATE TABLE my_unique_2 (
    ->   id INT UNSIGNED,
    ->   username VARCHAR(10),
    ->   UNIQUE(id),
    ->   UNIQUE(username)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

接着使用DESC查看my_unique_1表和my_unique_2表的结构，会发现两个表的结构是相同的，如下所示。

```
+----------+------------------+------+-----+---------+-------+
| Field    | Type             | Null | Key | Default | Extra |
+----------+------------------+------+-----+---------+-------+
| id       | int(10) unsigned | YES  | UNI | NULL    |       |
| username | varchar(10)      | YES  | UNI | NULL    |       |
+----------+------------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

在上述结果中，如果在id和username的Key列看到UNI，说明唯一约束已经添加成功，这两个字段是唯一键。值得一提的是，当表级约束仅建立在一个字段上时，其作用效果与列级约束相同。

(2) 为含唯一约束的字段插入记录，具体SQL语句及执行结果如下。

```
#① 插入不重复记录,插入成功
mysql> INSERT INTO my_unique_1 (id) VALUES(1);
Query OK, 1 rows affected (0.01 sec)
mysql> INSERT INTO my_unique_1 (id) VALUES(2);
```

```
Query OK, 1 rows affected (0.01 sec)
#② 插入重复记录,插入失败
mysql> INSERT INTO my_unique_1 (id) VALUES(1);
ERROR 1062 (23000): Duplicate entry '1' for key 'id'
#③ 查询插入的结果
mysql> SELECT * FROM my_unique_1;
+-------+----------+
| id    | username |
+-------+----------+
|   1   | NULL     |
|   2   | NULL     |
+-------+----------+
2 rows in set (0.00 sec)
```

从上述结果可以看出,添加唯一约束后,插入重复记录会失败。其中,username 字段出现了重复值 NULL,这是因为 MySQL 的唯一约束允许存在多个 NULL 值。

(3) 添加和删除唯一性约束。若为一个现有的表添加或删除唯一约束,无法通过修改字段属性的方式操作,而是按照索引的方式来操作。关于索引的概念和使用会在后面的章节中详细讲解,读者此时只需了解用到的这些操作即可。具体 SQL 语句及执行结果如下。

```
#① 创建测试表
mysql> CREATE TABLE my_unique_3 (id INT);
Query OK, 0 rows affected (0.01 sec)
#② 添加唯一约束
mysql> ALTER TABLE my_unique_3 ADD UNIQUE(id);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
#③ 查看添加结果
mysql> SHOW CREATE TABLE my_unique_3\G
*************** 1. row ***************
       Table: my_unique_3
Create Table: CREATE TABLE `my_unique_3` (
  `id` int(11) DEFAULT NULL,
UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
#④ 删除唯一约束
mysql> ALTER TABLE my_unique_3 DROP INDEX id;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
#⑤ 查看删除结果
mysql> SHOW CREATE TABLE my_unique_3\G
*************** 1. row ***************
       Table: my_unique_3
Create Table: CREATE TABLE `my_unique_3` (
  `id` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

在上述操作中,第③步的执行结果中出现了“UNIQUE KEY `id` (`id`)”,它是添加唯一

约束的完整语法，即 UNIQUE(id)的完整形式，如下所示。

```
UNIQUE KEY 索引名 (字段列表)
```

上述语法表示在添加唯一约束时创建索引，用于加快查询速度。其中，索引名可以自己指定，也可以省略，MySQL 会自动使用字段作为索引名。当需要对索引进行删除时，需要指定这个索引名。

(4) 创建复合唯一约束。在表级唯一性约束创建时，UNIQUE()的字段列表中，可以添加多个字段，组成复合唯一键，其特点是只有多个字段的值相同时才视为重复记录。具体 SQL 语句及执行结果如下。

```
#① 创建测试表，添加复合唯一键
mysql> CREATE TABLE my_unique_4 (
    ->    id INT UNSIGNED, username VARCHAR(10),
    ->    UNIQUE(id, username)
    -> );
Query OK, 0 rows affected (0.01 sec)
#② 插入不重复记录，插入成功
mysql> INSERT INTO my_unique_4 VALUES(1, '2');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO my_unique_4 VALUES(1, '3');
Query OK, 1 row affected (0.00 sec)
#③ 插入重复记录，插入失败
mysql> INSERT INTO my_unique_4 VALUES(1, '2');
ERROR 1062 (23000): Duplicate entry '1-2' for key 'id'
```

从上述结果可以看出，当同一个字段两次插入的记录相同时，插入成功，只有当两个字段同时发生重复时，插入记录失败。

3.2.4 主键约束

在 MySQL 中，为了快速查找表中的某条信息，可以通过设置主键来实现。主键可以唯一标识表中的记录，类似指纹、身份证用于标识人的身份一样。

主键约束通过 PRIMARY KEY 定义，它相当于唯一约束和非空约束的组合，要求被约束字段不允许重复，也不允许出现 NULL 值，每个表最多只允许含有一个主键。

主键约束的创建也分为列级和表级。其基本语法格式如下。

```
#列级约束
字段名 数据类型 PRIMARY KEY
#表级约束
PRIMARY KEY (字段名 1, 字段名 2, …)
```

在上述语法中，表级约束的字段若只有一个，则为单字段主键与列级约束添加的效果相同；若有多个，则为复合主键。复合主键需要用多个字段来确定一条记录的唯一性，类似于复合唯一键。

为了让读者更好地理解，下面通过案例演示主键约束的使用及注意事项。

(1) 创建 my_primary 表，为 id 字段添加主键约束。

```
mysql> CREATE TABLE my_primary (
    ->    id INT UNSIGNED PRIMARY KEY,
    ->    username VARCHAR(20)
    -> );
```

（2）使用 DESC 查看表结构，执行结果如下。

```
mysql> DESC my_primary;
+----------+------------------+------+-----+---------+-------+
| Field    | Type             | Null | Key | Default | Extra |
+----------+------------------+------+-----+---------+-------+
| id       | int(10) unsigned | NO   | PRI | NULL    |       |
| username | varchar(20)      | YES  |     | NULL    |       |
+----------+------------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

从上述结果可以看出，id 字段的 Key 列为 PRI，表示该字段为主键。同时，id 字段的 Null 列为 NO，表示该字段不能为 NULL。

（3）插入记录进行测试，具体 SQL 语句及执行结果如下。

```
#① 插入测试记录，插入成功
mysql> INSERT INTO my_primary VALUES(1, 'Tom');
Query OK, 1 row affected (0.00 sec)
#② 为主键插入 NULL 值，插入失败
mysql> INSERT INTO my_primary VALUES(NULL, 'Jack');
ERROR 1048 (23000): Column 'id' cannot be null
#③ 为主键插入重复值，插入失败
mysql> INSERT INTO my_primary VALUES(1, 'Alex');
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
```

从上述结果可以看出，添加主键约束后，插入重复值或 NULL 值会失败。

（4）为一个现有的表添加或删除主键约束，具体 SQL 语句及执行结果如下。

```
#① 删除主键约束
mysql> ALTER TABLE my_primary DROP PRIMARY KEY;
Query OK, 1 row affected (0.04 sec)
Records: 1  Duplicates: 0  Warnings: 0
#② 查看删除结果
mysql> DESC my_primary;
+----------+------------------+------+-----+---------+-------+
| Field    | Type             | Null | Key | Default | Extra |
+----------+------------------+------+-----+---------+-------+
| id       | int(10) unsigned | NO   |     | NULL    |       |
| username | varchar(20)      | YES  |     | NULL    |       |
+----------+------------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
#③ 删除 id 字段的非空约束（此步骤可选）
mysql> ALTER TABLE my_primary MODIFY id INT UNSIGNED;
Query OK, 0 rows affected (0.05 sec)
```

```
Records: 0  Duplicates: 0  Warnings: 0
#④ 添加主键约束
mysql>ALTER TABLE my_primary ADD PRIMARY KEY (id);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
#⑤ 查看添加结果
mysql>DESC my_primary;
+----------+-----------------+------+-----+---------+-------+
| Field    | Type            | Null | Key | Default | Extra |
+----------+-----------------+------+-----+---------+-------+
| id       | int(10) unsigned | NO  | PRI | NULL    |       |
| username | varchar(20)     | YES  |     | NULL    |       |
+----------+-----------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

从上述结果可以看出，在删除 id 字段的主键约束后，该字段的非空约束并没有被同时删除。若需要删除 id 字段的非空约束，执行第③步删除操作即可。

3.3 自动增长

在为数据表设置主键约束后，每次插入记录时，都需要检查主键的值，防止插入的值重复导致插入失败，这会给数据库的使用带来很多麻烦。为此，可以利用 MySQL 提供的自动增长功能来自动生成主键的值。

自动增长功能通过 AUTO_INCREMENT 来实现，其基本语法格式如下。

```
字段名 数据类型 AUTO_INCREMENT
```

在使用 AUTO_INCREMENT 时，需要注意以下 4 点。

(1) 一个表中只能有一个自动增长字段，该字段的数据类型是整数类型，且必须定义为键，如 UNIQUE KEY、PRIMARY KEY。

(2) 若为自动增长字段插入 NULL、0、DEFAULT 或在插入时省略该字段，则该字段就会使用自动增长值；若插入的是一个具体值，则不会使用自动增长值。

(3) 自动增长值从 1 开始自增，每次加 1。若插入的值大于自动增长的值，则下次插入的自动增长值会自动使用最大值加 1；若插入的值小于自动增长值，则不会对自动增长值产生影响。

(4) 使用 DELETE 删除记录时，自动增长值不会减小或填补空缺。

为了让读者更好地理解，下面通过案例演示自动增长的使用及注意事项。

(1) 创建 my_auto 表，为 id 字段添加自动增长。

```
mysql>CREATE TABLE my_auto (
    ->   id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    ->   username VARCHAR(20)
    ->);
```

（2）使用 DESC 查看表结构，执行结果如下。

```
mysql> DESC my_auto;
+-----------+------------------+-------+-----+---------+-----------------+
| Field     | Type             | Null  | Key | Default | Extra           |
+-----------+------------------+-------+-----+---------+-----------------+
| id        | int(10) unsigned | NO    | PRI | NULL    | auto_increment  |
| username  | varchar(20)      | YES   |     | NULL    |                 |
+-----------+------------------+-------+-----+---------+-----------------+
2 rows in set (0.00 sec)
```

（3）插入记录进行测试，具体 SQL 语句及执行结果如下。

```
#① 插入时省略 id 字段,将会使用自动增长值
mysql> INSERT INTO my_auto (username) VALUES('a');
Query OK, 1 row affected (0.00 sec)
#② 为 id 字段插入 NULL,将会使用自动增长值
mysql> INSERT INTO my_auto VALUES(NULL, 'b');
Query OK, 1 row affected (0.00 sec)
#③ 为 id 字段插入具体值 6
mysql> INSERT INTO my_auto VALUES(6, 'c');
Query OK, 1 row affected (0.00 sec)
#④ 为 id 字段插入 0,使用自动增长值
mysql> INSERT INTO my_auto VALUES(0, 'd');
Query OK, 1 row affected (0.00 sec)
```

（4）查看 my_auto 表中的数据，执行结果如下。

```
mysql> SELECT * FROM my_auto;
+----+----------+
| id | username |
+----+----------+
|  1 | a        |
|  2 | b        |
|  6 | c        |
|  7 | d        |
+----+----------+
4 rows in set (0.00 sec)
```

在上述结果中，最后一条记录的 id 字段在插入时使用了 0，MySQL 会忽略该值，使用自动增长值（即 id 最大值 6 进行加 1），从而得到 id 的值为 7。

（5）使用 SHOW CREATE TABLE 查看自动增长值，执行结果如下。

```
mysql> SHOW CREATE TABLE my_auto\G
*************************** 1. row ***************************
       Table: my_auto
Create Table: CREATE TABLE `my_auto` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(20) DEFAULT NULL,
```

```
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

在上述结果中,“AUTO_INCREMENT=8”用于指定下次插入的自动增长值为 8。若在下次插入时指定了大于 8 的值,此处的 8 会自动更新为下次插入值加 1。

(6) 为现有的表修改或删除自动增长,具体 SQL 语句及执行结果如下。

```
#① 修改自动增长值
mysql>ALTER TABLE my_auto AUTO_INCREMENT = 10;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
#② 删除自动增长
mysql>ALTER TABLE my_auto MODIFY id INT UNSIGNED;
Query OK, 5 rows affected (0.03 sec)
Records: 5  Duplicates: 0  Warnings: 0
#③ 重新为 id 添加自动增长
mysql>ALTER TABLE my_auto MODIFY id INT UNSIGNED AUTO_INCREMENT;
Query OK, 5 rows affected (0.03 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

需要注意的是,在为字段删除自动增长并重新添加自动增长后,自动增长的初始值会自动设为该列现有的最大值加 1。在修改自动增长值时,修改的值若小于该列现有的最大值,则修改不会生效。

小提示: 通过“SHOW VARIABLES LIKE 'auto_increment%';”可以查看 MySQL 中用于维护自动增长的变量分别是 auto_increment_increment(默认值 1)和 auto_increment_offset(默认值为 1),通过更改这两个变量可以改变自动增长的计算方式,读者可以参考官方手册中的详细说明。

3.4 字符集与校对集

MySQL 默认使用的字符集为 latin1(ISO-8859-1 的别名),这是一种单字节编码的字符集。由于中文汉字属于多字节编码的字符,若要保存中文,需要设置成其他支持中文的字符集,如 GBK、UTF8。校对集是字符之间的比较关系,在比较、排序等操作时都会用到校对集。本节将针对 MySQL 中的字符集与校对集进行详细讲解。

3.4.1 字符集与校对集概述

1. 字符集

字符(character)指计算机中保存的各种文字和符号,包括各种国家的文字、标点符号、图形符号、数字等。由于计算机采用二进制保存数据,用户输入的字符将会按照一定的规则转换为二进制后保存,这个过程就是字符编码(character encoding),将一系列字符的编码规则组合起来就形成了字符集(character set,charset)。

在计算机的发展历史中,出现了许多字符集。MySQL 也提供了各种字符集的支持,通

过“SHOW CHARACTER SET;”可以查看可用字符集，如图 3-1 所示。



```
命令提示符 - mysql  -uroot
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | cp1252 West European        | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |      1 |
| ascii    | US ASCII                    | ascii_general_ci    |      1 |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |      3 |
```

图 3-1　MySQL 可用字符集

图 3-1 显示了字符集名称（Charset）、描述信息（Description）、默认校对集（Default collation）和单字符的最大长度（Maxlen）。

常用字符集有 latin1、gbk、utf8，具体说明如表 3-9 所示。

表 3-9　常用字符集说明

字　符　集	单字符最大长度	支持的语言
latin1	1 字节	西欧字符、希腊字符等
gbk	2 字节	简体和繁体中文、日文、韩文等
utf8	3 字节	世界上大部分国家的文字

从表 3-9 中可以看出，单字符占用的存储空间越多，所支持的语言越多。另外，表中列举的字符集都向下兼容 ASCII（美国信息交换标准代码），若输入的字符在 ASCII 范围内（0x00～0x7F），则编码结果是完全相同的。

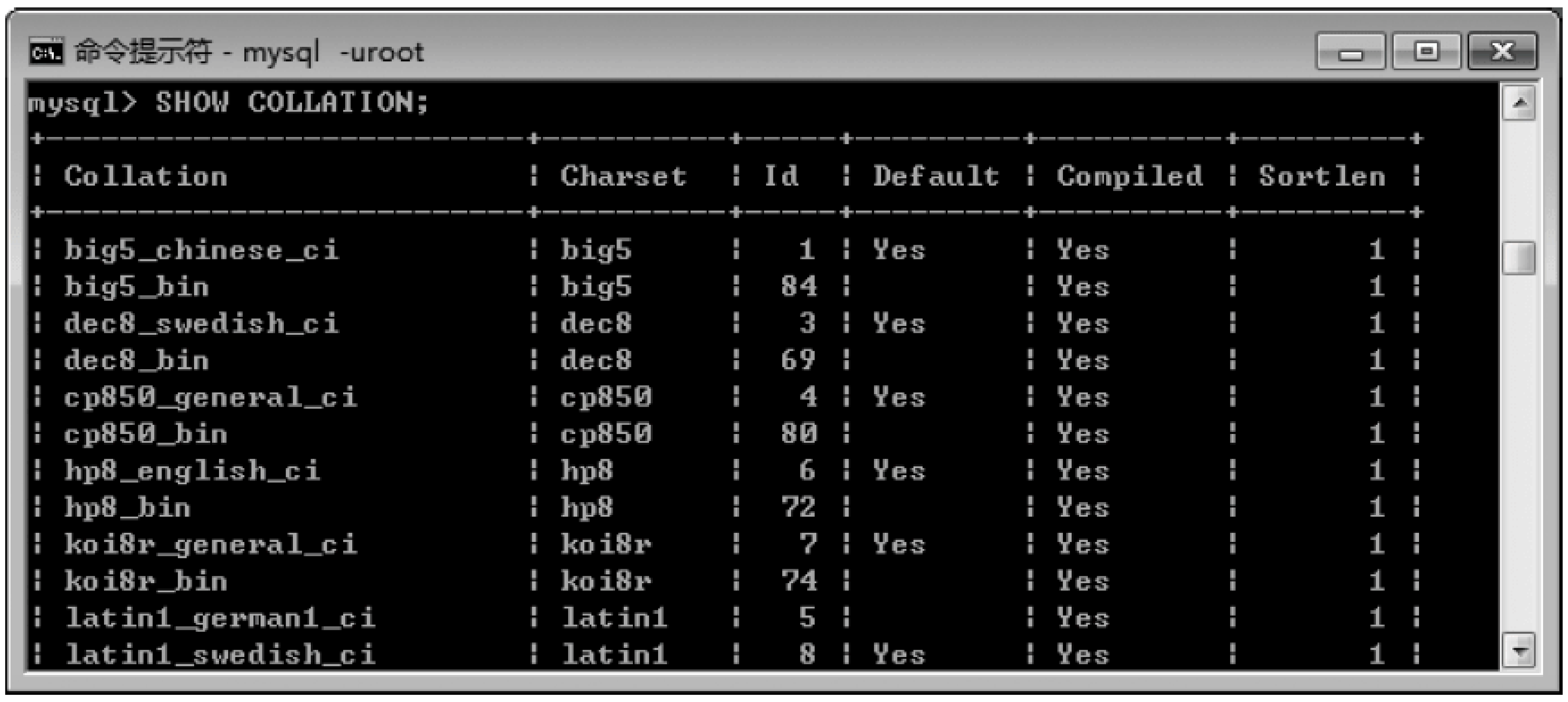
脚下留心

由于历史原因，MySQL 中的 utf8 编码与标准的 UTF-8（RFC 3629）存在一些差别。标准的 UTF-8 规定一个字符最多使用 4 字节，而 MySQL 中的 utf8 编码一个字符最多使用 3 字节，这导致 UTF-8 中的一些特殊字符（如 emoji 符号）无法在 MySQL 的 utf8 编码中使用。为了解决这个问题，MySQL 从 5.5.3 版本开始新增了 utf8mb4 编码，将一个字符扩展到 4 字节。因此，如果要考虑支持 RFC 3629 规范，应使用 utf8mb4 编码。

2. 校对集

MySQL 中提供了许多校对集，用于为不同字符集指定比较和排序规则。例如，latin1 字符集默认的校对集为 latin1_swedish_ci。校对集的名称由“_”分隔的 3 部分组成，开头是对应的字符集，中间是国家名或 general，结尾是 ci、cs 或 bin。其中，ci 表示不区分大小写，cs 表示区分大小写，bin 表示以二进制方式比较。

通过“SHOW COLLATION;”可以查看 MySQL 可用的校对集，如图 3-2 所示。



```
命令提示符 - mysql -uroot
mysql> SHOW COLLATION;
+--------------------------+---------+----+---------+----------+---------+
| Collation                | Charset | Id | Default | Compiled | Sortlen |
+--------------------------+---------+----+---------+----------+---------+
| big5_chinese_ci          | big5    |  1 | Yes     | Yes      |       1 |
| big5_bin                 | big5    | 84 |         | Yes      |       1 |
| dec8_swedish_ci          | dec8    |  3 | Yes     | Yes      |       1 |
| dec8_bin                 | dec8    | 69 |         | Yes      |       1 |
| cp850_general_ci         | cp850   |  4 | Yes     | Yes      |       1 |
| cp850_bin                | cp850   | 80 |         | Yes      |       1 |
| hp8_english_ci           | hp8     |  6 | Yes     | Yes      |       1 |
| hp8_bin                  | hp8     | 72 |         | Yes      |       1 |
| koi8r_general_ci         | koi8r   |  7 | Yes     | Yes      |       1 |
| koi8r_bin                | koi8r   | 74 |         | Yes      |       1 |
| latin1_german1_ci        | latin1  |  5 |         | Yes      |       1 |
| latin1_swedish_ci        | latin1  |  8 | Yes     | Yes      |       1 |
```

图 3-2　MySQL 可用校对集

图 3-2 显示了校对集名称(Collation)、对应的字符集(Charset)、校对集 ID(Id)、是否为对应字符集的默认校对集(Default)、是否已编译(Compiled)以及排序的内存需求量(Sortlen)。

3.4.2　字符集与校对集的设置

根据不同的需求,字符集与校对集的设置分为 4 个方面,分别是 MySQL 环境、数据库、数据表以及字段。下面分别进行讲解。

1. MySQL 环境

使用 MySQL 命令“SHOW VARIABLES LIKE 'character%';”可以查看与字符集相关的变量,输出结果如下所示。

```
mysql> SHOW VARIABLES LIKE 'character% ';
+--------------------------+------------------------------+
| Variable_name            | Value                        |
+--------------------------+------------------------------+
| character_set_client     | gbk                          |
| character_set_connection | gbk                          |
| character_set_database   | latin1                       |
| character_set_filesystem | binary                       |
| character_set_results    | gbk                          |
| character_set_server     | latin1                       |
| character_set_system     | utf8                         |
| character_sets_dir       | C:\mysql5.7\share\charsets\  |
+--------------------------+------------------------------+
8 rows in set, 1 warning (0.01 sec)
```

上述结果表示当前会话(Session)使用的字符集,在不同客户端环境中的输出结果可能不同。这里所说的会话,是指从客户端登录服务器到退出的整个过程。例如,依次打开两个客户端并登录服务器,就产生了两个会话,不同客户端属于不同的会话。由此可见,不同的客户端可以指定不同的字符集环境配置,服务器会按照不同的配置进行处理。

接下来通过表 3-10 对输出结果中的变量名进行详细说明。

表 3-10　与字符集相关的变量

变　量　名	说　　明
character_set_client	客户端字符集
character_set_connection	客户端与服务器连接使用的字符集
character_set_database	默认数据库使用的字符集(从 5.7.6 版本开始不推荐使用)
character_set_filesystem	文件系统字符集
character_set_results	将查询结果(如结果集或错误消息)返回给客户端的字符集
character_set_server	服务器默认字符集
character_set_system	服务器用来存储标识符的字符集
character_sets_dir	安装字符集的目录

在表 3-10 中,读者应重点关注的变量是 character_set_client、character_set_connection、character_set_results 和 character_set_server,具体解释如下。

(1) character_set_server 决定了新创建的数据库默认使用的字符集。需要注意的是,数据库的字符集决定了数据表的默认字符集,数据表的字符集决定了字段的默认字符集。由于 character_set_server 的值为 latin1,因此在前面的学习中,创建的数据库、数据表和字段的默认字符集都是 latin1。

(2) character_set_client、character_set_connection 和 character_set_results 分别对应客户端、连接层和查询结果的字符集。通常情况下,这 3 个变量的值是相同的,具体值由客户端的编码而定,从而使客户端输入的字符和输出的查询结果都不会出现乱码。

通过"set 变量名 = 值;"命令可以更改变量的值,示例命令如下。

```
set character_set_client=gbk;
set character_set_connection=gbk;
set character_set_results=gbk;
```

由于上述命令输入比较麻烦,在 MySQL 中还可以通过"set names 字符集;"直接修改上述 3 个变量的值,即"set names gbk;"。

小提示:

(1) 使用 set 或 set names 修改字符集只对当前会话有效,不影响其他会话,且会话结束后,下次会话仍然使用默认值。

(2) 与 character_set_connection、character_set_database、character_set_server 对应的校对集分别通过变量 collation_connection、collation_database、collation_server 来指定。

(3) 若字段使用 utf8 字符集,而客户端使用 gbk 字符集,MySQL 会自动进行编码转换。由于 utf8 和 gbk 本质上是不同的字符集,虽然大部分常见的字符可以转换成功,但若遇到其中一个字符集中没有的特殊字符,则可能会出现乱码。

2. 数据库

在创建数据库时设定字符集和校对集的语法格式如下。

```
[DEFAULT] CHARACTER SET [=] charset_name
[DEFAULT] COLLATE [=] collation_name
```

在上述语法中，CHARACTER SET用于指定字符集，COLLATE用于指定校对集。若仅指定字符集，表示使用该字符集的默认校对集；若仅指定校对集，表示使用该校对集对应的字符集。示例SQL语句如下。

```
#创建数据库,指定字符集为 utf8,使用默认校对集 utf8_general_ci
CREATE DATABASE mydb_1 CHARACTER SET utf8;
#创建数据库,指定字符集为 utf8,校对集为 utf8_bin
CREATE DATABASE mydb_2 CHARACTER SET utf8 COLLATE utf8_bin;
```

3. 数据表

数据表的字符集与校对集在表选项中设定，语法格式如下。

```
[DEFAULT] CHARACTER SET [=] charset_name
[DEFAULT] COLLATE [=] collation_name
```

上述语法与指定数据库字符集的语法类似。若没有为数据表指定字符集，则自动使用数据库的字符集。示例SQL语句如下。

```
CREATE TABLE my_charset (
  username VARCHAR(20)
) CHARACTER SET utf8 COLLATE utf8_bin;
```

上述SQL语句指定数据表字符集为utf8，校对集为utf8_bin。

小提示：在MySQL中，还可以将CHARACTER SET简写为CHARSET。例如，通过“SHOW CREATE TABLE my_charset;”可以看到设置字符集和校对集的语法形式为DEFAULT CHARSET=utf8 COLLATE=utf8_bin。

4. 字段

字段的字符集与校对集在字段属性中设定，语法格式如下。

```
[CHARACTER SET charset_name] [COLLATE collation_name]
```

在上述语法中，若没有为字段设定字符集与校对集，则会自动使用数据表的字符集与校对集。示例SQL语句如下。

```
CREATE TABLE my_charset (
  username VARCHAR(20) CHARACTER SET utf8 COLLATE utf8_bin
);
```

上述SQL语句指定username字段的字符集为utf8，校对集为utf8_bin。

3.5 动手实践：设计用户表

数据库的学习在于多看、多学、多思考、多动手，只有将理论与实际相结合，才能够体会数据库开发与管理的重要性，展现知识学习的价值与力量。接下来请结合本章所学的知识

完成用户表的设计。

【实践目标】

本实践的目标就是能够根据文字提示，完成对用户表的设计，为表中的字段设置合理的数据类型、约束以及字符集。

【实践需求】

在电子商务网站中，提供了用户注册功能，当用户在注册表单中填写信息后，提交表单，就可以注册一个新用户。为了保存用户的数据，需要在数据库中创建一张用户表，该表需要保存的用户信息如下。

- 用户名：可以使用中文，不允许重复，长度在 20 个字符以内。
- 手机号码：长度为 11 个字符。
- 性别：有男、女、保密 3 种选择。
- 注册时间：注册时的日期和时间。
- 会员等级：表示会员等级的数字，最高为 100。

【动手实践】

1. 创建用户表

根据需求创建用户表，为每个字段设置合理的数据类型。具体 SQL 语句如下。

```
mysql> CREATE TABLE mydb.user (
    ->    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '用户 id',
    ->    username VARCHAR(20) UNIQUE NOT NULL COMMENT '用户名',
    ->    mobile CHAR(11) NOT NULL COMMENT '手机号码',
    ->    gender ENUM('男', '女', '保密') NOT NULL COMMENT '性别',
    ->    reg_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP COMMENT '注册时间',
    ->    level TINYINT UNSIGNED NOT NULL COMMENT '会员等级'
    -> ) DEFAULT CHARSET=utf8;
Query OK, 0 rows affected (0.01 sec)
```

从上述 SQL 语句可以看出，用户表的名称为 user，表中有 6 个字段，字符集为 utf8。

2. 添加测试记录

创建用户表后，添加测试记录，具体 SQL 语句如下。

```
mysql> INSERT INTO mydb.user VALUES (
    ->    NULL, '小明',                      #用户 id、用户名
    ->    '12311111111', '男',               #手机号码、性别
    ->    '2018-01-01 11:11:11', 1           #注册时间、会员等级
    -> );
Query OK, 1 row affected (0.00 sec)
```

在上述 SQL 语句中，添加了用户名为小明的用户，用户 id 为 NULL 表示使用自动增

长值。

3. 查询用户表中的记录

查看用户表中的记录，具体 SQL 语句及执行结果如下。

```
mysql> SELECT * FROM mydb.user;
+----+----------+-------------+--------+---------------------+-------+
| id | username | mobile      | gender | reg_time            | level |
+----+----------+-------------+--------+---------------------+-------+
|  1 | 小明     | 12311111111 | 男     | 2018-01-01 11:11:11 |     1 |
+----+----------+-------------+--------+---------------------+-------+
1 row in set (0.00 sec)
```

从上述结果可以看出，user 表中只有一个用户，即小明。读者也可以尝试添加更多的用户。

3.6　本章小结

本章主要讲解了常用的数据类型、表的约束、自动增长，以及字符集和校对集。这些内容十分零碎，但又非常重要，需要通过实践练习加以透彻理解。数据类型与约束是本章的重点，希望读者掌握每种数据类型和约束的适用场景，可以结合表的实际情况去运用。

3.7　课后练习

一、填空题

1. MySQL 数据类型中存储整数数值并且占用字节数最小的是________。
2. 数据表中字段的唯一性约束是通过________关键字定义的。
3. 设置数据表的字段值自动增加使用________属性。
4. 在创建表时不允许某列为空，则可以使用________约束。
5. 使用 INT 类型保存数字 1 占用的字节数为________。

二、判断题

1. 一个数据表中可以定义多个主键。(　　)
2. 一个数据表中可以定义多个非空字段。(　　)
3. 非空约束指的是字段的值不能为空字符串。(　　)
4. TEXT 类型存储的最大字节数为 65535。(　　)
5. ENUM 类型的数据只能从枚举列表中取，并且只能取一个。(　　)

三、选择题

1. 下列选项中，用于存储整数数值的是(　　)。

A. FLOAT　　B. DOUBLE　　C. MEDIUMINT　D. VARCHAR

2. 下列选项中，适合存储文章内容或评论的数据类型是(　　)。

A. CHAR　　B. VARCHAR　　C. TEXT　　D. VARBINARY

3. 下列选项中，表示日期和时间的数据类型是(　　)。

A. DECIMAL(6, 2)　　B. DATE

C. YEAR　　D. TIMESTAMP

4. 下面关于 DECIMAL(6, 2)的说法中，正确的是(　　)。

A. 它不可以存储小数

B. 6 表示数据的长度，2 表示小数点后的长度

C. 6 表示最多的整数位数，2 表示小数点后的长度

D. 总共允许最多存储 8 位数字

5. 下列关于主键的说法中，正确的是(　　)。

A. 主键允许为 NULL 值

B. 主键允许有重复值

C. 主键必须来自于另一个表中的值

D. 主键具有非空性，唯一性

四、简答题

1. 请简述 ENUM 和 SET 数据类型的区别。

2. 请简述 CHAR、VARCHAR 和 TEXT 数据类型的区别。

五、实训题

1. 请设计一张学生表，选择合理的数据类型保存学号、姓名、性别、出生日期、入学日期、家庭住址信息。

2. 请设计一张留言表，用于保存网站留言板中游客发表的留言。

第 4 章 数据库设计

学习目标

- 熟悉数据库设计的基本步骤
- 掌握数据库设计范式的使用
- 掌握电子商务网站的数据库设计

第 2、3 章主要讲解了 MySQL 的常用操作，相信读者已经能够灵活使用 SQL 语句对数据进行操作了。但是，在将数据库技术应用到实际需求时，仅仅掌握数据库的使用还不够，还需要研究如何设计一个合理、规范和高效的数据库。本章将围绕数据库设计进行详细讲解。

4.1 数据库设计概述

数据库设计要求设计人员对数据库有深入的理解，才能设计出高质量的数据库。数据库设计一般分为 6 个阶段，分别是需求分析、概念数据库设计、逻辑数据库设计、物理数据库设计、数据库实施、数据库运行和维护。

1. 需求分析

在需求分析阶段，数据库设计人员需要分析用户的需求，将分析结果记录下来，形成需求分析报告。在这个阶段中，双方需要进行深入的沟通，以避免理解不准确导致后续的工作出现问题。在需求分析中有许多琐碎、耗时的工作，常见的工作如下。

(1) 收集数据。一个企业内的数据可能分散、零碎，由不同人员负责管理。为了使用数据库系统管理这些数据，需要尽可能多地收集数据，并理解企业的业务过程和数据处理流程，理解数据处理的性能需求。可以利用数据流图等工具辅助分析与理解。

(2) 解决冲突，包括命名冲突（同名异义、异名同义）、属性冲突、结构冲突。例如，商品库存数量是否包含已下订单未出库数量；到货数量和入库数量以哪一个为准；用户名和昵称、真实姓名如何区分；性别使用男、女，还是 0、1 或 f、m 来表示。

(3) 为数据形成一些标准，如商品编号一共有多少位，未来是否会增加位数，每一位的含义是什么；订单编号按照什么规则生成，如何避免编号重复，编号中包含哪些信息，是否加入一些随机数防止被推测等。

2. 概念数据库设计

在概念数据库设计阶段，将对用户的需求进行综合、归纳、抽象，形成概念模型。概念模型使设计人员摆脱数据库系统的具体技术问题，将精力集中在分析数据及数据之间联系等方面。一般通过绘制 E-R 图，直观呈现数据库设计人员对用户需求的理解。

3. 逻辑数据库设计

逻辑数据库设计面向数据库系统，在概念数据库设计中完成 E-R 图等成果后，将其转换为 DBMS 支持的数据模型(如关系模型)，完成实体、属性和联系的转换。

在进行逻辑数据库设计时，应遵循一些规范化理论，如范式(将在后面的小节中详细讲解)。不规范的设计可能会导致数据库出现大量冗余、插入异常、删除异常等问题。

4. 物理数据库设计

物理数据库设计阶段需要确定数据库的存储结构、文件类型等。通常 DBMS 为了保证其独立性与可移植性，承担了大部分任务，数据库设计人员只需要考虑硬件、操作系统的特性，为数据表选择合适的存储引擎，为字段选择合适的数据类型等，以及评估磁盘空间需求等工作。

5. 数据库实施

数据库实施就是将前面那些工作的成果实施起来，比如使用 SQL 语句创建数据库、数据表，编写与调试应用程序等。

6. 数据库运行和维护

数据库运行和维护就是将数据库系统正式投入运行，在运行后进行一些维护、调整、备份、升级等工作。

4.2　数据库设计范式

数据库设计对数据的存储性能、数据的操作都有很大的关系。为了避免不规范的数据库出现数据冗余，造成插入、删除、更新操作异常等情况，就要满足一定的规范化要求，这就是范式(Normal Form)。

根据要求的程度不同，范式有多种级别，最常用的有第一范式(1NF)、第二范式(2NF)和第三范式(3NF)，它们由 Edgar Frank Codd 于 1971 年相继提出。后来，又有人提出了 Boyce-Codd 范式(BCNF)、第四范式(4NF)和第五范式(5NF)等。一般来说，数据库设计只需满足第三范式(3NF)就可以了，下面对前三范式进行详细讲解。

1. 第一范式(1NF)

第一范式(1NF)是指数据库表的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值，或不能有重复的属性。简而言之，第一范式遵

从原子性,属性不可再分。

为了使读者更好地理解,下面通过表 4-1 和表 4-2 演示不满足第一范式的情况(表格中的列标题“编号”表示主键,用下划线标注)。

表 4-1 用户联系方式表

编号	联系方式
1	张三 邮箱:zhangsan@example.com,手机号:18900000000
2	李四 邮箱:lisi@example.com,手机号:15900000000、17300000000

表 4-2 用户联系方式表

编号	用户名	邮箱	手机号	手机号
1	张三	zhangsan@example.com	18900000000	
2	李四	lisi@example.com	15900000000	17300000000

表 4-1 的问题在于“联系方式”包含了多个值,可以再细分;表 4-2 的问题在于“手机号”属性重复。

为了满足第一范式,应将用户和联系方式分成两个表保存,两个表是一对多的联系。具体如表 4-3 和表 4-4 所示。

表 4-3 用户表

用户编号	用户名
1	张三
2	李四

表 4-4 联系方式表

编号	用户编号	联系方式	具体值
1	1	邮箱	zhangsan@example.com
2	1	手机号	18900000000
3	2	邮箱	lisi@example.com
4	2	手机号	15900000000
5	2	手机号	17300000000

通过表 4-3 和表 4-4 可以看出,无论一个用户有多少个联系方式,都可以使用这两张表来保存。

2. 第二范式(2NF)

第二范式(2NF)是在第一范式的基础上建立起来的,满足第二范式必须先满足第一范式。第二范式要求实体的属性完全依赖于主键,不能仅依赖主键的一部分(对于复合主键而言)。简而言之,第二范式遵从唯一性,非主键字段需完全依赖主键。

为了使读者更好地理解，下面通过表 4-5 和表 4-6 演示不满足第二范式的情况。

表 4-5　订单表

订单编号	订单商品	购买件数	下单时间
1	铅笔	3	2019-01-20 8:30:15
2	钢笔	2	2019-01-20 8:30:15
3	圆珠笔	1	2019-02-12 9:20:16

表 4-6　用户表

用户编号	订单编号	用户名	付款状态
1	1	张三	已支付
1	2	张三	未支付
2	3	李四	已支付

在表 4-6 中，用户编号和订单编号组成了复合主键，付款状态完全依赖复合主键，而用户名只依赖用户编号。

采用上述方式设计的用户表存在如下问题。

- 插入异常：如果一个用户没有下过订单，则该用户无法插入；
- 删除异常：如果删除一个用户所有的订单，则该用户也会被删除；
- 更新异常：由于用户名冗余，修改一个用户时需要修改多条记录。如果稍有不慎，漏改某些记录，会出现更新异常。

为了满足第二范式，将复合主键移动到订单表中，如表 4-7 和表 4-8 所示。

表 4-7　用户表

用户编号	用户名
1	张三
2	李四

表 4-8　订单表

订单编号	用户编号	订单商品	购买件数	下单时间	付款状态
1	1	铅笔	3	2019-01-20 8:30:15	已支付
2	1	钢笔	2	2019-01-20 8:30:15	未支付
3	2	圆珠笔	1	2019-02-12 9:20:16	已支付

3. 第三范式（3NF）

第三范式（3NF）是在第二范式的基础上建立起来的，即满足第三范式必须先满足第二范式。第三范式要求一个数据表中每一列数据都和主键直接相关，而不能间接相关。简而言之，第三范式就是非主键字段不能相互依赖。

为了使读者更好地理解，下面通过表4-9演示不满足第三范式的情况。

表4-9 用户表

用户编号	用户名	用户等级	享受折扣
1	张三	1	0.95
2	李四	1	0.95
3	王五	2	0.85

在表4-9中，用户享受的折扣与用户等级相关，两者存在依赖关系。采用这种方式设计的用户表存在如下问题。

- 插入异常：新插入用户的等级如果在1、2之外，其享受的折扣无处参考；
- 删除异常：如果删除某个等级下所有的用户，该等级对应的折扣也被删除；
- 更新异常：如果修改某个用户的等级，折扣也必须随之修改；如果修改某个等级的折扣，又因为折扣存在冗余，容易发生漏改。

为了满足第三范式，将等级与折扣拆分到单独的表中，如表4-10和表4-11所示。

表4-10 用户表

用户编号	用户名	用户等级
1	张三	1
2	李四	1
3	王五	2

表4-11 折扣表

用户等级	享受折扣
1	0.95
2	0.85

多学一招：函数依赖

函数依赖(Functional Dependency)是由数学派生的术语，是数据依赖的一种类型。它表示根据一个属性(或属性集)的值，可以找到另一个属性(或属性集)的值。例如，将商品关系模式中的属性“商品id”设为X，属性集(商品名称，商品价格)设为Y，根据X可以找到Y，说明X决定了Y，Y函数依赖于X，记为X→Y。

函数依赖根据依赖属性的不同可以分为完全函数依赖、部分函数依赖和传递函数依赖，具体如下。

(1) 完全函数依赖。以表4-8为例，使用订单编号和用户编号可以决定付款状态，而若只有订单编号，无法决定是哪一个用户创建了订单，只有用户编号，无法决定是哪一个订单。因此，付款状态这个属性函数依赖(订单编号，用户编号)属性集。

(2) 部分函数依赖。以表4-6为例，用户名依赖用户编号，但不依赖订单编号，因此用户名这个属性部分函数依赖(订单编号，用户编号)属性集。

(3) 传递函数依赖。以表 4-9 为例,享受折扣依赖用户等级,用户等级依赖用户编号,所以享受折扣传递函数依赖用户编号。

由此可见,在第一范式限定了一个关系模式的所有属性都是不可分的基本数据项以后,第二范式消除了部分函数依赖,第三范式消除了传递函数依赖。

多学一招:反范式

反范式是一种逆规范化设计,其目的主要是为了提高查询效率。范式虽然减少了数据冗余,但是增加了表的数量,这会使查询变得复杂,尤其是连接多张表查询数据时,会使查询性能降低。例如,商品销量可以通过查询订单表中的购买记录计算出来,当需要查询大量商品的销量时,就需要花费许多时间去计算销量。

为了提高查询效率,不必每次查询计算销量,可以违反范式的要求,在商品表中增加一个销量字段,当商品被购买时就增加销量字段。这种方式的缺点是容易出现数据不一致的问题。例如,用户购买商品后,程序出现意外没有增加销量,销量数据就有误了。

在实际开发中,若选择采取反范式的设计,应该提前评估其可能出现的问题,并准备一套解决方案。例如,通过存储过程进行操作,定期检查数据的一致性等。

4.3　数据建模工具

在数据库设计过程中,对于业务复杂、修改频繁的场合,手工绘制 E-R 图显得非常低效,这时可以利用数据建模工具提高效率。常见的数据建模工具有 ERwin Data Modeler、Power Designer、MySQL Workbench 等。其中 MySQL Workbench 由 MySQL 官方出品,具有开源和商业两个版本,支持 Windows 和 Linux 系统,下面介绍该工具的使用。

(1) 打开 MySQL Workbench 6.3 CE,执行菜单栏【File】→【New Model】命令创建新模型,如图 4-1 所示。

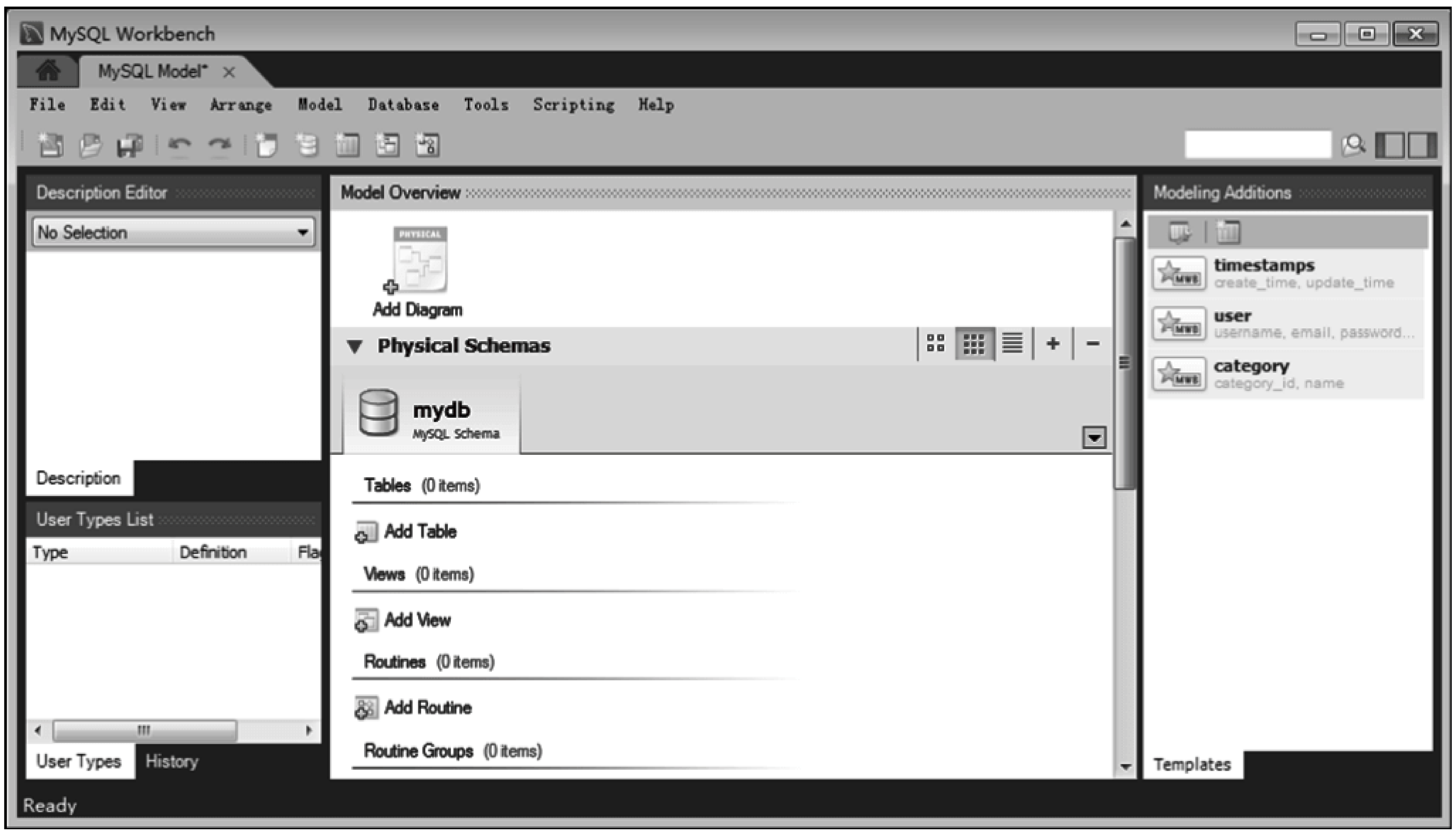


图 4-1　创建模型

从图 4-1 中可以看出，当前打开了 MySQL Model 选项卡，在该选项卡中可以绘制 EER 图或编辑物理模式(Physical Schemas)。其中，EER 图是指增强版的 E-R 图，更加适合专业人员进行数据建模，在绘制完成后可以直接转换成 SQL。

(2) 双击【Add Diagram】按钮(或执行菜单栏【Model】→【Add Diagram】命令)可以添加 EER 图。EER 图绘制区域的基本说明如图 4-2 所示。

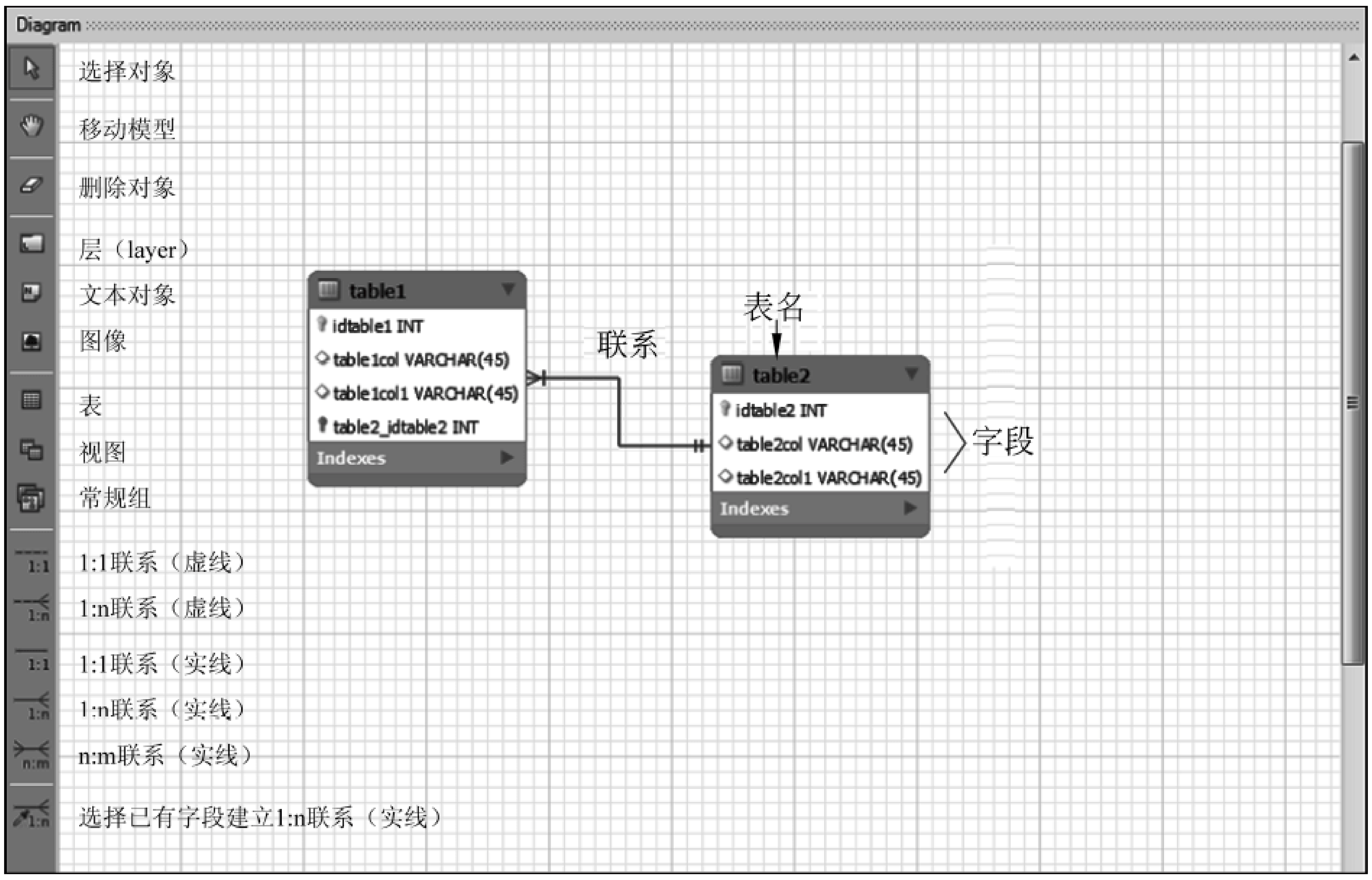


图 4-2 EER 图绘制区域

在图 4-2 中，table1 和 table2 是通过“表”按钮绘制的两张表。单击“表”按钮后，在画布的空白区域单击即可绘制出一个新表。

(3) 双击绘制出来的 table1 或 table2，会出现一个面板，如图 4-3 所示。

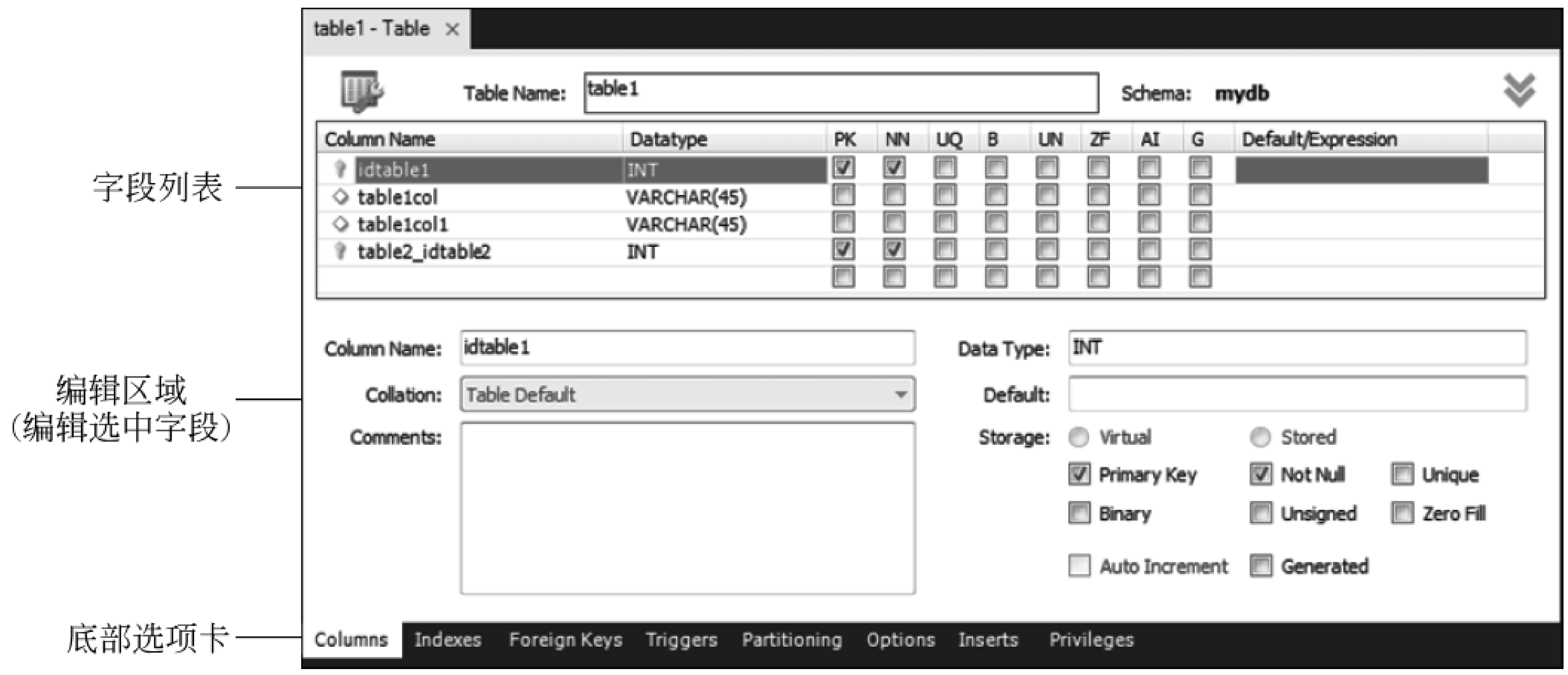


图 4-3 编辑字段

在图 4-3 所示的面板中，可以方便地对表中的字段、数据类型、约束等内容进行编辑。双击字段列表最后一行(空白行)可以添加字段，单击已有字段可以进行编辑。字段列表中

的 PK、NN 等复选框与编辑区域 Storage 中的 Primary Key、Not Null 对应。

另外，面板底部的选项卡可以在列（Columns）、索引（Indexes）、外键（Foreign Keys）、触发器（Triggers）等功能之间切换。

(4) 完成 EER 图的编辑后，执行菜单栏【File】→【Export】→【Forward Engineer SQL CREATE Script】命令导出 SQL。生成的 SQL 预览结果如图 4-4 所示。

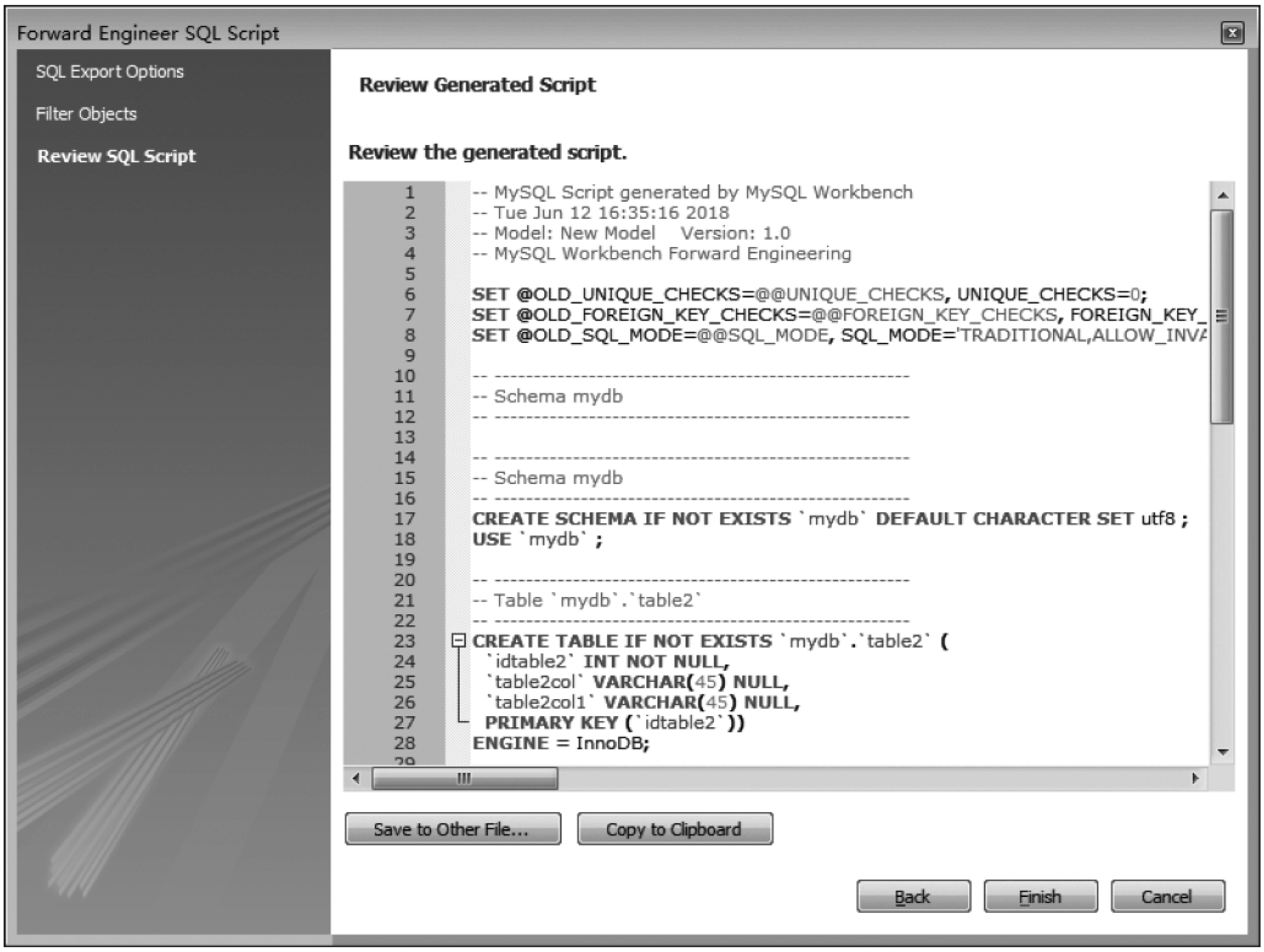


图 4-4　SQL 预览结果

从图 4-4 中可以看出，相比用户手动编写的 SQL，软件自动生成的 SQL 会比较复杂，它的语法非常严谨，并加上了一些 SET 环境配置，以防止不同环境差异导致 SQL 无法按照预期执行。

4.4　数据库设计——电子商务网站

在了解数据库设计的基本流程、规范以后，为了学以致用，本节将以电子商务网站为例，演示如何进行数据库设计。通过实际操作，加深对数据库设计的理解。

4.4.1　需求分析

电子商务网站是一种在互联网上开展电子商务的平台，常见的电子商务网站有淘宝、天猫、京东、亚马逊等。一般来说，电子商务网站最基本的功能是在线购物，它分为前台和后台，后台面向网站的运营人员，用于录入数据，前台面向访问网站的用户，用于选购商品。下面通过表 4-12 列举电子商务网站中的一些常见需求。

表 4-12 电子商务网站需求说明

模块	需　求	说　明
商品	商品 *	商品基本信息，如商品名称、商品描述等
	商品分类 *	如手机、书籍、服装等，支持多级分类
	商品属性 *	商品分类的属性信息，如手机品牌、屏幕尺寸等
	商品规格 *	如颜色、尺寸，不同规格的价格、库存数不同
	商品评论 *	用户发表的商品评论
	商品评分	用户对商品的评分，范围 1～5 分
	商品图片	后台上传的图片路径
	商品咨询	用户发表的商品咨询信息
用户	用户 *	注册用户，可以购买商品、编辑个人资料等
	收货地址	用户的收货地址表
	购物车	用户保存在购物车中的商品
	用户等级	用户等级划分
	用户订阅	用户订阅的邮箱电子杂志
	用户收藏	用户收藏的商品
	用户日志	登录记录、充值记录、消费记录、提现记录等日志信息
订单	订单	用户购买商品下的订单
	订单日志	等待确认、已确认、请求取消、已取消、已发货等日志
	发货	发货单，如收货地址、收货人、物流单号等
	售后服务	申请退换货、退款等
文章	文章	文章标题、所属分类、文章内容等
	文章分类	如新闻动态、网站公告、帮助信息等，支持多级分类
公共	系统设置	网站设置，如网站标题、LOGO、备案号、客服 QQ 等
	配送地区	前台用户收货地址的地区选择范围，如省、市、区、县等
	链接	页面顶部、底部、主导航、搜索区域的链接
	广告	各种类型的广告信息
	推荐商品	在首页、列表页等页面显示的推荐商品
	运行日志	用户注册、登录失败、提交订单、取消订单等产生的日志
后台	后台用户	保存后台用户的数据
	后台用户组	后台用户组，如管理员、编辑、仓管员、客服等
	后台权限	除管理员外，为其他用户组开放的权限
	后台日志	后台的操作日志，如商品发布日志、商品修改日志等

表 4-12 中列举的需求非常多,但对于初学者来说,此时并不需要花费大量的时间研究这些商业领域的需求,最重要的是掌握解决问题的方法。因此,本节仅选取一些典型需求(带有 * 标注)进行详细讲解。有余力的读者可以尝试解决其他需求。

由于电子商务行业竞争激烈、需求变化快,往往对网站的开发时间要求比较严格,希望网站能尽早上线。对于这样的需求,可以采用迭代式开发,先追求小而精,将核心功能优先完成,为后期的扩展预留空间,然后再通过版本迭代逐渐发展成大而全。

在设计数据库时,考虑到需求将来可能会发生变化的情况,应设计一个尽量能够适应需求变化的数据库,其关键点如下。

(1) 避免数据经常发生变化。将数据保存到数据库中后,数据是不会变的,只有用户对数据进行修改时,数据才会发生变化。在设计数据库时,应尽量保存不变的数据,以减少修改数据带来的开销。例如,在设计用户表时一般会保存出生日期而不是年龄,这是因为利用出生日期和当前日期可以推算出年龄,而若保存年龄,数据每一年都需要发生一次改变。

在实际开发中,经常会遇到一些数据被大量访问,而该数据必须经过复杂的运算才能够获得的情况。例如,在网站首页显示销量排名前 10 的商品,需要统计出所有商品的销量并降序排列,然后取出前 10 条记录。当有大量用户同时访问首页时,就会给数据库带来极大的负担,导致页面打开缓慢。为此,可以利用 Memcached、Redis 等技术对热门数据进行缓存,以减少数据库的查询次数。读者可以查询相关资料了解这些技术。

(2) 避免经常修改表结构。表结构的修改包括字段的增减、数据类型和约束的修改等。对于已经上线的项目,修改表结构可能会导致服务暂停,如果有更好的方案,则尽量避免修改表结构。例如,在网站首页中增加一个显示推荐商品的功能时,不必在商品表中增加一个“推荐”字段,而是专门创建一个“推荐商品”表。其目的是将功能数据与商品数据分离。原因是商品数据一般不会发生变化,而功能数据往往会随着功能的增加或减少而经常发生变化。“推荐商品”只是一个附加功能,应尽可能减少其对商品数据产生的影响。

(3) 尽可能详细地记录日志。利用日志记录数据的变化,可以为将来的功能增加提供数据基础。例如,当网站用户的账号被盗时,为了验明真实情况,需要确定账号被盗取的时间、登录的 IP 地址,以及充值、消费、订单等数据是否发生过变化。若没有记录日志,则难以追溯。

(4) 避免删除数据。对于已经上线的网站,删除数据是非常敏感的操作,很有可能带来一些始料未及的后果。例如,用户因误操作删除了订单,希望恢复订单,需要在项目中增加订单回收站功能。若数据库中的订单记录已被删除,即使增加了回收站功能,订单也无法恢复。因此,在设计数据库时,应考虑数据既可以被删除,又可以被恢复。例如,在订单表中增加一个“是否删除”的字段,用来标识记录是否被删除。当查询订单时,通过 WHERE 获取未被删除的订单,当查询订单回收站时,通过 WHERE 获取已删除的订单。

4.4.2 准备工作

在 MySQL 中创建一个 shop 数据库,用来保存电子商务网站中的数据。在创建数据库后,使用 USE shop 选择数据库,具体 SQL 语句如下。

```
mysql> CREATE DATABASE shop;
Query OK, 1 row affected (0.00 sec)
mysql> USE shop;
Database changed
```

在对数据表命名时，为了更好地区分项目，为所有的数据表加上“sh_”前缀。例如，商品表命名为“sh_goods”。

小提示： 在实际开发中，使用各种编程语言编写应用程序访问数据库时，最常见的问题是没有对用户输入的数据进行处理，直接使用字符串拼接到 SQL 中，这样会导致用户输入的一些特殊字符(如单引号、反斜线、百分号等)被解析，破坏了原有 SQL 的语义，出现 SQL 注入等安全问题。因此，在编写应用程序时，一定要对特殊字符进行转义，或者借助一些库或框架来完成处理。

4.4.3　商品分类表

商品分类表用于保存分类名称、分类排序、是否显示等信息，并要求支持如图 4-5 所示的多级分类嵌套。

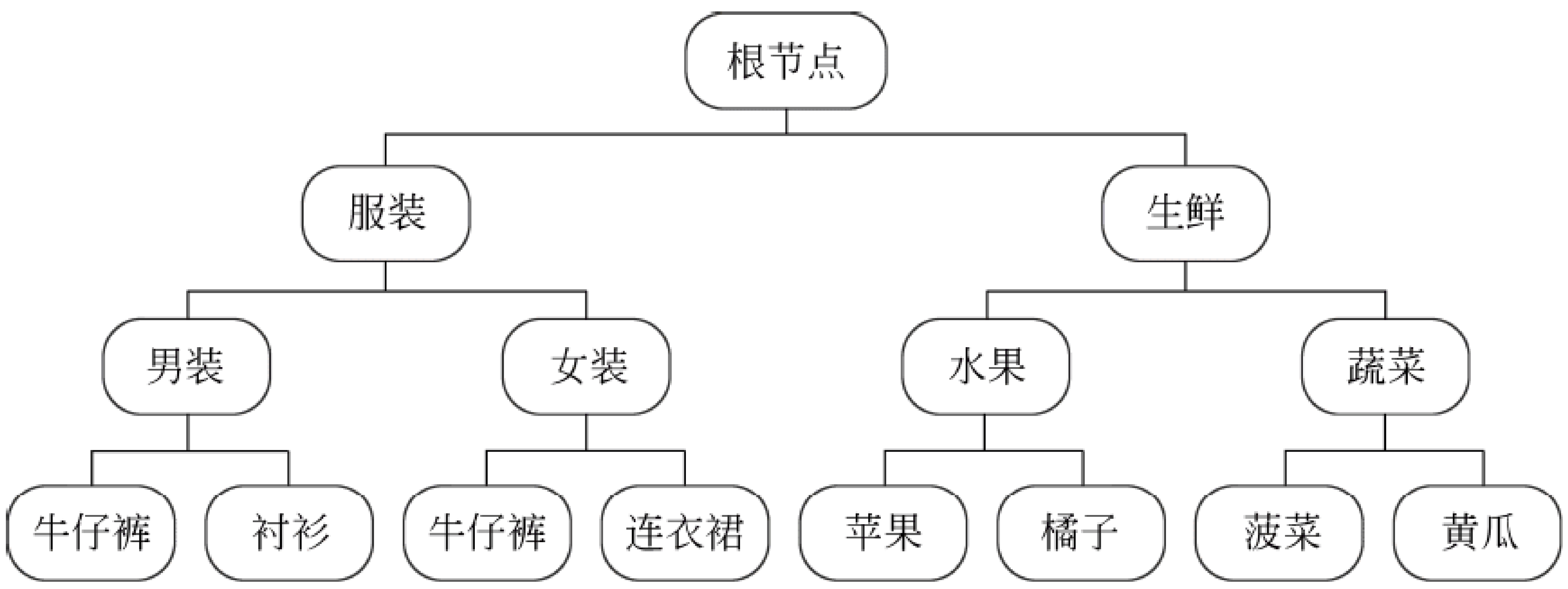


图 4-5　分类结构图

在图 4-5 中，商品分类是树形结构，父分类和子分类是一对多的关系。接下来通过表 4-13 展示商品分类表的具体结构。

表 4-13　商品分类表(sh_goods_category)

字　段　名	数据类型和约束	说　　明
id	INT UNSIGNED PRIMARY KEY AUTO_INCREMENT	分类 id
parent_id	INT UNSIGNED NOT NULL DEFAULT 0	上级分类 id
name	VARCHAR(100) NOT NULL DEFAULT''	名称
sort	INT NOT NULL DEFAULT 0	排序
is_show	TINYINT UNSIGNED NOT NULL DEFAULT 0	是否显示
create_time	DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP	创建时间
update_time	DATETIME DEFAULT NULL	更新时间

在表 4-13 中设计的商品分类表 sh_goods_category 共有 7 个字段，具体说明如下。

- 分类 id：主键，使用无符号整型存储，自动增长。
- 上级分类 id：保存上级分类的 id，若值为 0，表示上级分类是根节点。
- 名称：长度在 100 个字符以内。
- 排序：用于对同级分类进行排序，按照从小到大的顺序排列。
- 是否显示：0 表示不显示，1 表示显示。
- 创建时间：该分类被插入时的时间，后台使用。
- 更新时间：该分类的最后一次修改的时间，后台使用。

小提示：

（1）对于 is_show 字段，其数据类型选择 TINYINT 而不是 ENUM，这是考虑到需求将来有可能会发生变化，ENMU 类型在增加枚举值时需要修改表结构，而 TINYINT 类型不需要修改表结构。

（2）如果父分类的 is_show 值为 0，则该分类和它的子分类都不会显示。子分类必须满足自身 is_show 和父分类 is_show 都为 1 时才会显示。

（3）update_time 字段的默认为 NULL，这是因为在新插入某一条记录时，该记录并没有发生过修改操作，所以用 NULL 来表示。

下面根据表 4-13 所示的表结构，创建商品分类表，具体 SQL 语句如下。

```
CREATE TABLE sh_goods_category (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '分类 id',
  parent_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '上级分类 id',
  name VARCHAR(100) NOT NULL DEFAULT '' COMMENT '名称',
  sort INT NOT NULL DEFAULT 0 COMMENT '排序',
  is_show TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '是否显示',
  create_time DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  update_time DATETIME DEFAULT NULL COMMENT '更新时间'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

商品分类表创建完成后，添加测试数据，具体 SQL 语句如下。

```
INSERT INTO sh_goods_category (id, parent_id, name) VALUES
(1, 0, '办公'), (2, 1, '耗材'), (3, 2, '文具'),
(4, 0, '电子产品'), (5, 4, '通讯'), (6, 5, '手机'),
(7, 4, '影音'), (8, 7, '音箱'), (9, 7, '耳机'),
(10, 4, '电脑'), (11, 10, '台式电脑'), (12, 10, '笔记本'),
(13, 0, '服装'), (14, 13, '女装'), (15, 14, '风衣'), (16, 14, '毛衣');
```

在上述 SQL 语句中，测试数据一共包含 16 条分类记录，层次关系如下。

```
办公-耗材-文具            电子产品-通讯-手机          电子产品-影音-音箱
电子产品-影音-耳机        电子产品-电脑-台式电脑      电子产品-电脑-笔记本
服装-女装-风衣            服装-女装-毛衣
```

4.4.4　商品表

商品分类与商品是一对多的联系，一个分类中有多件商品。商品表的具体结构如表 4-14

所示。

表 4-14 商品表(sh_goods)

字 段 名	数据类型和约束	说 明
id	INT UNSIGNED PRIMARY KEY AUTO_INCREMENT	商品 id
category_id	INT UNSIGNED NOT NULL DEFAULT 0	分类 id
spu_id	INT UNSIGNED NOT NULL DEFAULT 0	SPU id
sn	VARCHAR(20) NOT NULL DEFAULT ''	编号
name	VARCHAR(120) NOT NULL DEFAULT ''	名称
keyword	VARCHAR(255) NOT NULL DEFAULT ''	关键词
picture	VARCHAR(255) NOT NULL DEFAULT ''	图片
tips	VARCHAR(255) NOT NULL DEFAULT ''	提示
description	VARCHAR(255) NOT NULL DEFAULT ''	描述
content	TEXT NOT NULL	详情
price	DECIMAL(10,2) UNSIGNED NOT NULL DEFAULT 0	价格
stock	INT UNSIGNED NOT NULL DEFAULT 0	库存
score	DECIMAL(3,2) UNSIGNED NOT NULL DEFAULT 0	评分
is_on_sale	TINYINT UNSIGNED NOT NULL DEFAULT 0	是否上架
is_del	TINYINT UNSIGNED NOT NULL DEFAULT 0	是否删除
is_free_shipping	TINYINT UNSIGNED NOT NULL DEFAULT 0	是否包邮
sell_count	INT UNSIGNED NOT NULL DEFAULT 0	销量计数
comment_count	INT UNSIGNED NOT NULL DEFAULT 0	评论计数
on_sale_time	DATETIME DEFAULT NULL	上架时间
create_time	DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP	创建时间
update_time	DATETIME DEFAULT NULL	更新时间

在表 4-14 中设计的商品表 sh_goods 共有 21 个字段,具体说明如下。

- 商品 id:主键,使用无符号整型存储,自动增长。
- 分类 id:商品所属分类,使用无符号整型存储,自动增长。
- SPU id:标准化产品单元 id(具体会在后面讲解)。
- 编号:长度在 20 个字符以内。
- 名称:长度在 120 个字符以内。
- 关键词:用于在搜索和浏览列表时更准确地找到该商品。
- 图片:用于在商品列表中显示商品的预览图,保存图片路径。
- 提示:用于醒目地显示该商品的提示信息,如促销信息、推荐语等。
- 描述:用于通过简短的文字描述商品的基本信息。
- 详情:用于描述商品的详细信息。

- 评分：用户购买商品后的评分(1～5)，此处为平均分，主要用于排序。
- 价格、库存：商品的价格和库存。
- 是否上架、是否删除、是否包邮：0 表示是，1 表示否。
- 销量计数、评论计数：商品的销量和评论数量，主要用于排序。
- 上架时间：商品的上架时间，主要用于排序。
- 创建时间：该商品被插入时的时间，后台使用。
- 更新时间：该商品最后一次修改的时间，后台使用。

小提示：

(1) 对于商品编号，若没有特殊要求，一般情况下使用商品 id 作为编号。但考虑到将来有可能更换另外一套规则来管理编号，因此这里预留了一个编号字段。

(2) 由于商品分类是多级嵌套结构，一般来说，商品所属的分类应该是叶子节点的分类。如果现有分类不能满足需求，可以增加一个名称为"其他"的分类。

(3) 用户上传的图片等文件，一般是由 Web 服务器来管理的，数据库中只保存它们的引用路径。

下面根据表 4-14 所示的表结构，创建商品表，对应的 SQL 语句如下。

```
CREATE TABLE sh_goods (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '商品 id',
  category_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '分类 id',
  spu_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT 'SPU id',
  sn VARCHAR(20) NOT NULL DEFAULT '' COMMENT '编号',
  name VARCHAR(120) NOT NULL DEFAULT '' COMMENT '名称',
  keyword VARCHAR(255) NOT NULL DEFAULT '' COMMENT '关键词',
  picture VARCHAR(255) NOT NULL DEFAULT '' COMMENT '图片',
  tips VARCHAR(255) NOT NULL DEFAULT '' COMMENT '提示',
  description VARCHAR(255) NOT NULL DEFAULT '' COMMENT '描述',
  content TEXT NOT NULL COMMENT '详情',
  price DECIMAL(10, 2) UNSIGNED NOT NULL DEFAULT 0 COMMENT '价格',
  stock INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '库存',
  score DECIMAL(3, 2) UNSIGNED NOT NULL DEFAULT 0 COMMENT '评分',
  is_on_sale TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '是否上架',
  is_del TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '是否删除',
  is_free_shipping TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '是否包邮',
  sell_count INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '销量计数',
  comment_count INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '评论计数',
  on_sale_time INT DATETIME DEFAULT NULL COMMENT '上架时间',
  create_time DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  update_time DATETIME DEFAULT NULL COMMENT '更新时间'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

创建商品表后，插入测试数据，具体 SQL 语句如下。

```
INSERT INTO sh_goods (id, category_id, name, keyword, content, price,
stock, score, comment_count) VALUES
(1, 3, '2B 铅笔', '文具', '考试专用', 0.5, 500, 4.9, 40000),
(2, 3, '钢笔', '文具', '练字必不可少', 15, 300, 3.9, 500),
```

```
(3, 3, '碳素笔', '文具', '平时使用', 1, 500, 5, 98000),
(4, 12, '超薄笔记本', '电子产品', '轻小便携', 5999, 0, 2.5, 200),
(5, 6, '智能手机', '电子产品', '人人必备', 1999, 0, 5, 98000),
(6, 8, '桌面音箱', '电子产品', '扩音装备', 69, 750, 4.5, 1000),
(7, 9, '头戴耳机', '电子产品', '独享个人世界', 109, 0, 3.9, 500),
(8, 10, '办公电脑', '电子产品', '适合办公', 2000, 0, 4.8, 6000),
(9, 15, '收腰风衣', '服装', '春节潮流单品', 299, 0, 4.9, 40000),
(10, 16, '薄毛衣', '服装', '居家旅行必备', 48, 0, 4.8, 98000);
```

4.4.5 商品规格表

在电子商务网站中有 SPU(Standard Product Unit,标准化产品单元)和 SKU(Stock Keeping Unit,库存量单位)的概念。它们可以简单理解为看待商品的两种不同的角度,SPU 从信息聚合的角度看待商品,SKU 从库存管理的角度看待商品。

例如,一款饮料有 500ml 和 750ml 两种规格,对应的库存和价格不同,它们是不同的 SKU,在 sh_goods 表中每个 SKU 商品是一条商品记录。在店铺中展示商品时,如果按照 SKU 进行展示,会出现大量冗余的信息(同一款饮料因为规格不同重复出现),不利于顾客挑选商品,因此店铺通常会将相同款式的商品合并成 SPU,如图 4-6 所示。



图 4-6 商品 SPU 与 SKU

在图 4-6 中,整个页面可以理解为一件 SPU 商品,在页面中可以切换 500ml 和 750ml 这两种规格的 SKU 商品,当切换后,价格和库存也会发生变化。

需要注意的是,图 4-6 演示的商品规格只有一个维度,即容量,而有些商品的规格可能有多个维度,如手机的规格可以分为网络、颜色、内存 3 个维度,如图 4-7 所示。

由此可见,不同商品的规格维度不固定,不同维度的规格项的数量也不固定。为了满足这种需求,一般需要如下 4 张表才能够实现。

```
sh_goods_spu(id, name)
sh_goods_spec(id, name)
sh_goods_spec_item(id, spec_id, name, description, picture)
sh_goods_spec_set(goods_id, spec_item_id)
```

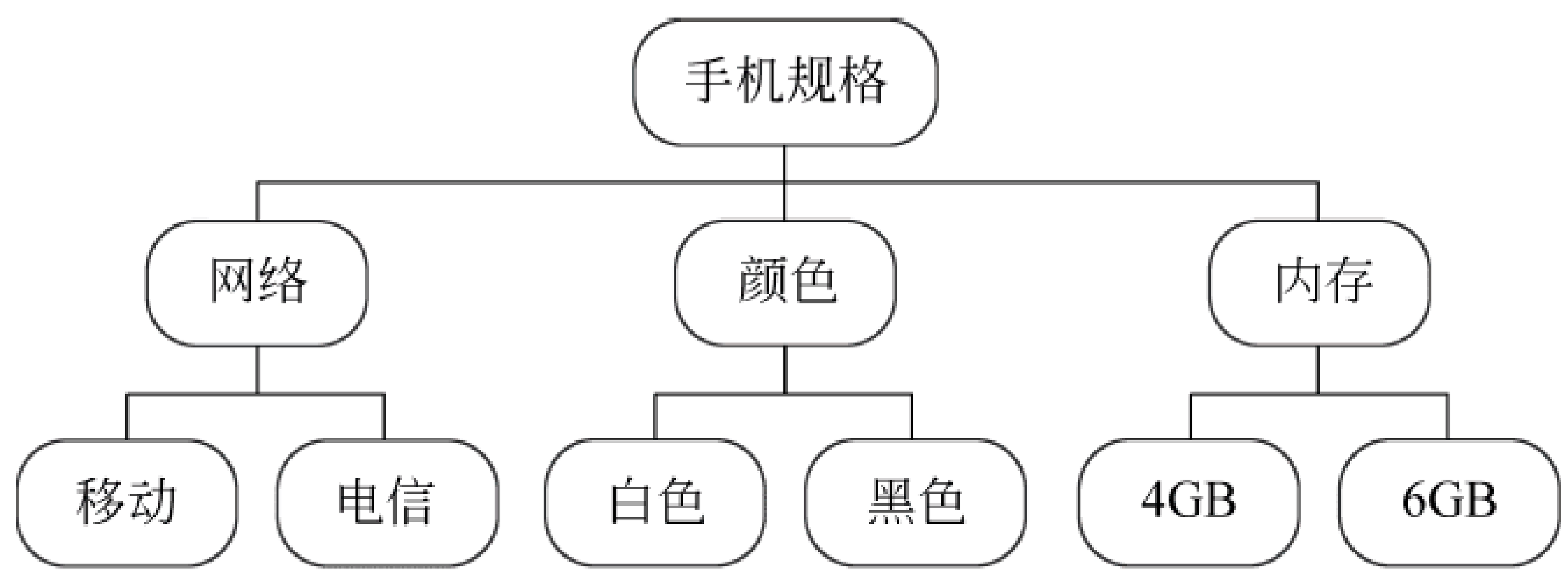


图 4-7　手机的规格

在上述表中，sh_goods_spu 保存 SPU 的 id 和名称，一个 SPU 包含多件商品，sh_goods 中的 spu_id 字段就表示该商品所属的 SPU；sh_goods_spec 保存规格，如"网络、颜色、内存"；sh_goods_spec_item 保存规格项，如"移动、电信"，一个规格包含多个规格项；sh_goods_spec_set 保存商品的规格项组合，如"电信-白色-4GB"（实际保存的是 id）。

接下来通过表 4-15～表 4-18 展示上述 4 张表的表结构。

表 4-15　商品 SPU 表（sh_goods_spu）

字　段　名	数据类型和约束	说　　明
id	INT UNSIGNED PRIMARY KEY AUTO_INCREMENT	SPU id
name	VARCHAR(80) NOT NULL DEFAULT ''	SPU 名称

表 4-16　商品规格表（sh_goods_spec）

字　段　名	数据类型和约束	说　　明
id	INT UNSIGNED PRIMARY KEY AUTO_INCREMENT	规格 id
name	VARCHAR(80) NOT NULL DEFAULT ''	规格名称

表 4-17　商品规格项表（sh_goods_spec_item）

字　段　名	数据类型和约束	说　　明
id	INT UNSIGNED PRIMARY KEY AUTO_INCREMENT	规格项 id
spec_id	INT UNSIGNED NOT NULL DEFAULT 0	规格 id
name	VARCHAR(80) NOT NULL DEFAULT ''	名称
description	VARCHAR(255) NOT NULL DEFAULT ''	描述
picture	VARCHAR(255) NOT NULL DEFAULT ''	可选图

表 4-18　商品规格组合表（sh_goods_spec_set）

字　段　名	数据类型和约束	说　　明
goods_id	INT UNSIGNED NOT NULL DEFAULT 0	SKU id
spec_item_id	INT UNSIGNED NOT NULL DEFAULT 0	规格 id

下面根据表 4-15～表 4-18 所示的表结构，创建上述 4 张表，对应的 SQL 语句如下。

```
#商品 SPU 表
CREATE TABLE sh_goods_spu (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT 'SPU id',
  name VARCHAR(80) NOT NULL DEFAULT '' COMMENT 'SPU 名称'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
#商品规格表
CREATE TABLE sh_goods_spec (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '规格 id',
  name VARCHAR(80) NOT NULL DEFAULT '' COMMENT '规格名称'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
#商品规格项表
CREATE TABLE sh_goods_spec_item (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '规格项 id',
  spec_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '规格 id',
  name VARCHAR(80) NOT NULL DEFAULT '' COMMENT '名称',
  description VARCHAR(255) NOT NULL DEFAULT '' COMMENT '描述',
  picture VARCHAR(255) NOT NULL DEFAULT '' COMMENT '可选图'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
#商品规格组合表
CREATE TABLE sh_goods_spec_set (
  goods_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT 'SKU id',
  spec_item_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '规格 id'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

数据表创建完成后，为上述 4 张表添加测试数据，具体 SQL 语句如下。

```
INSERT INTO sh_goods_spu (id, name) VALUES
(1, '新款智能手机');
INSERT INTO sh_goods_spec (id, name) VALUES
(1, '网络'), (2, '颜色'), (3, '内存');
INSERT INTO sh_goods_spec_item (id, spec_id, name) VALUES
(1, 1, '移动'), (2, 1, '电信'), (3, 2, '白色'),
(4, 2, '黑色'), (5, 3, '4GB'), (6, 3, '6GB');
INSERT INTO sh_goods_spec_set (goods_id, spec_item_id) VALUES
(5, 2), (5, 3), (5, 5);
```

在上述测试数据中，goods_id 为 5 的商品(智能手机)的规格项有“电信”“白色”和“4GB”，表示该商品的规格为“网络：电信，颜色：白色，内存：4GB”，该商品和其他规格的商品组成了一个 SPU，即 sh_goods_spu 表中 id 为 1 的“新款智能手机”。

需要注意的是，在查询商品列表时，如果直接查询 sh_goods 表中的记录，会导致一个 SPU 中的多个 SKU 商品都显示在列表中，占用列表的空间，不利于挑选商品。为了解决这个问题，可以按照表中的 spu_id 字段去除重复记录，只保留一条商品记录即可。

小提示：在实际应用中，考虑到多张表的查询操作过于复杂，有时会通过逆规范化的设计增加一些冗余字段以方便查询。例如，在 sh_goods_spu 表中增加一个字段，将每个 SPU 相关的规格、规格项、规格组合以 JSON 格式编码后保存，从而简化查询。

4.4.6　商品属性表

商品属性主要有两个作用，一个是在商品展示页面中显示商品属性表格，一个是在商品列表页面中筛选商品，分别如图 4-8 和图 4-9 所示。

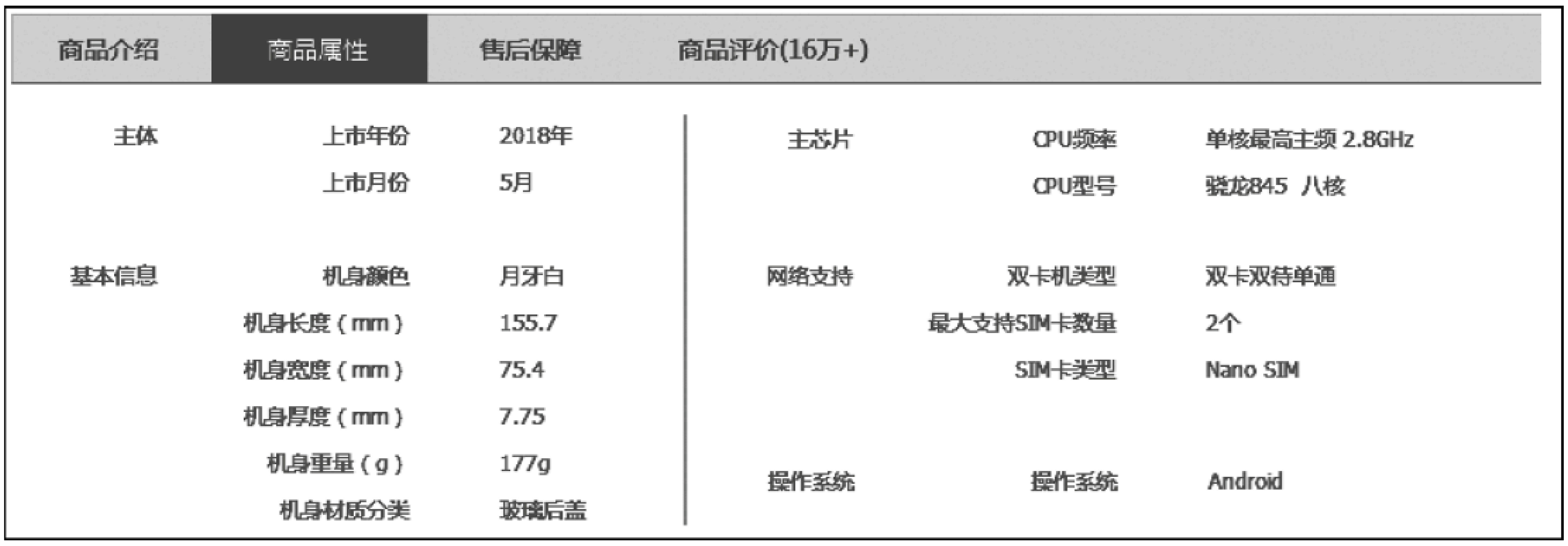


图 4-8　商品属性

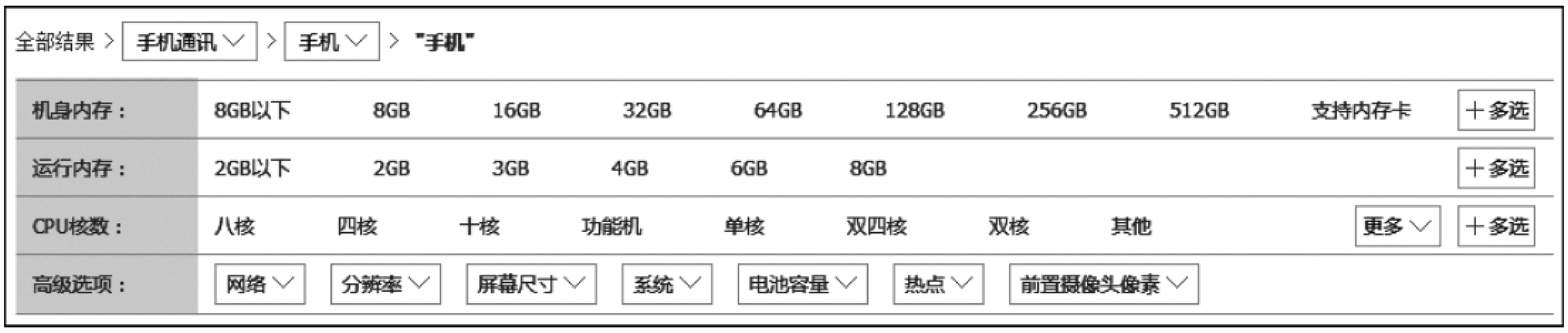


图 4-9　商品筛选

需要注意的是，图 4-8 和图 4-9 中的属性是针对手机分类的商品设计的，而本项目有多种商品分类，应为不同分类的商品设计不同的属性模板。

为了满足上述需求，可以通过如下 4 张表来实现。

```
sh_goods_attr(id, parent_id, category_id, name, sort)
sh_goods_attr_value(id, goods_id, attr_id, attr_value)
sh_goods_selector(id, parent_id, category_id, name, sort)
sh_goods_selector_value(id, goods_id, selector_id, selector_value)
```

在上述表中，sh_goods_attr 保存商品属性，一般有两个层级，第 1 级如“主体、基本信息、主芯片”，第 2 级如“上市年份、上市月份、机身颜色”；sh_goods_attr_value 保存商品的属性值；sh_goods_selector 保存筛选条件，一般有 2～3 级，2 级如“机身内存－8GB 以下”，3 级如“高级选项-网络-电信”；sh_goods_selector_value 保存商品的筛选值。

严格来说，sh_goods_attr 和 sh_goods_selector 表中的 category_id 字段存在冗余，因为商品属性存在层级关系，当通过 category_id 查找商品属性时，首先找到父属性，然后通过父属性找到子属性，此时子属性的 category_id 是没有意义的，它的值和父属性的 category_id 字段相同。若要消除冗余，应将父属性和子属性拆分成两张表，但这里为了简化操作，没有进行拆分，在实际开发中应避免这个冗余字段出现问题。

接下来通过表 4-19～表 4-22 展示上述 4 张表的表结构。

表 4-19 商品属性表(sh_goods_attr)

字 段 名	数据类型和约束	说 明
id	INT UNSIGNED PRIMARY KEY AUTO_INCREMENT	属性 id
parent_id	INT UNSIGNED NOT NULL DEFAULT 0	上级属性 id
category_id	INT UNSIGNED NOT NULL DEFAULT 0	商品分类 id
name	VARCHAR(80) NOT NULL DEFAULT ''	名称
sort	INT NOT NULL DEFAULT 0	排序

表 4-20 商品属性值表(sh_goods_attr_value)

字 段 名	数据类型和约束	说 明
id	INT UNSIGNED PRIMARY KEY AUTO_INCREMENT	属性值 id
goods_id	INT UNSIGNED NOT NULL DEFAULT 0	商品 id
attr_id	INT UNSIGNED NOT NULL DEFAULT 0	属性 id
attr_value	VARCHAR(80) NOT NULL DEFAULT ''	属性值

表 4-21 商品筛选表(sh_goods_selector)

字 段 名	数据类型和约束	说 明
id	INT UNSIGNED PRIMARY KEY AUTO_INCREMENT	筛选 id
parent_id	INT UNSIGNED NOT NULL DEFAULT 0	上级筛选 id
category_id	INT UNSIGNED NOT NULL DEFAULT 0	商品分类 id
name	VARCHAR(80) NOT NULL DEFAULT ''	名称
sort	INT NOT NULL DEFAULT 0	排序

表 4-22 商品筛选值表(sh_goods_selector_value)

字 段 名	数据类型和约束	说 明
id	INT UNSIGNED PRIMARY KEY AUTO_INCREMENT	筛选值 id
goods_id	INT UNSIGNED NOT NULL DEFAULT 0	商品 id
selector_id	INT UNSIGNED NOT NULL DEFAULT 0	筛选 id
selector_value	VARCHAR(80) NOT NULL DEFAULT ''	筛选值

下面根据表 4-19～表 4-22 所示的表结构，创建上述 4 张表对应的 SQL 语句如下。

```
#商品属性表
CREATE TABLE sh_goods_attr (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '属性 id',
  parent_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '上级属性 id',
```

```
  category_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '商品分类 id',
  name VARCHAR(80) NOT NULL DEFAULT '' COMMENT '名称',
  sort INT NOT NULL DEFAULT 0 COMMENT '排序'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
#商品属性值表
CREATE TABLE sh_goods_attr_value (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '属性值 id',
  goods_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '商品 id',
  attr_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '属性 id',
  attr_value VARCHAR(80) NOT NULL DEFAULT '' COMMENT '属性值'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
#商品筛选表
CREATE TABLE sh_goods_selector (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '筛选 id',
  parent_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '上级筛选 id',
  category_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '商品分类 id',
  name VARCHAR(80) NOT NULL DEFAULT '' COMMENT '名称',
  sort INT NOT NULL DEFAULT 0 COMMENT '排序'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
#商品筛选值表
CREATE TABLE sh_goods_selector_value (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '筛选值 id',
  goods_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '商品 id',
  selector_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '筛选 id',
  selector_value VARCHAR(80) NOT NULL DEFAULT '' COMMENT '筛选值'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

完成数据表的创建后，为上述 4 张表添加测试数据，具体 SQL 语句如下。

```
INSERT INTO sh_goods_attr VALUES
(1, 0, 6, '基本信息', 0), (2, 1, 6, '机身颜色', 0),
(3, 1, 6, '输入方式', 1), (4, 1, 6, '操作系统', 2),
(5, 0, 6, '屏幕', 1), (6, 5, 6, '屏幕尺寸', 0), (7, 5, 6, '屏幕材质', 1),
(8, 5, 6, '分辨率', 2), (9, 0, 6, '摄像头', 2),
(10, 9, 6, '前置摄像头', 0), (11, 9, 6, '后置摄像头', 1),
(12, 0, 6, '电池信息', 3), (13, 12, 6, '电池容量', 0),
(14, 12, 6, '是否可拆卸', 1);
INSERT INTO sh_goods_attr_value VALUES
(1, 5, 2, '黑色'), (2, 5, 3, '触摸屏'), (3, 5, 4, 'Android'),
(4, 5, 6, '5.5寸'), (5, 5, 7, 'IPS'), (6, 5, 8, '1920*1080'),
(7, 5, 10, '1600万'), (8, 5, 11, '800万'),
(9, 5, 13, '3500mAh'), (10, 5, 14, '否');
```

小提示：若希望筛选范围是一个区间(如手机的存储容量有 1GB、2GB、4GB 等选择)，有两种实现思路，一种是使用描述范围(如“2GB 以上”“1GB～2GB”)作为筛选项，另一种是在 sh_goods_selector_value 表中增加一个数字类型的字段，专门保存数字类型的属性值，在查询时使用大于、小于运算符来表示一个区间。

4.4.7 用户表

由于网站分为前后台，这里所说的用户是指前台用户，也就是网站的访客通过注册账号成为网站的用户。一般来说，数据库中的用户表包含两类信息，一类是登录信息，即账号和密码，账号通常是编号、用户名、邮箱或手机号等具有唯一性的值；另一类是用户的个人信息，如姓名、性别、出生年月、所在地等。

接下来通过表 4-23 展示用户表的具体结构。

表 4-23 用户表（sh_user）

字　段　名	数据类型和约束	说　　明
id	INT UNSIGNED PRIMARY KEY AUTO_INCREMENT	用户 id
name	VARCHAR(100) NOT NULL UNIQUE DEFAULT "	用户名
password	VARCHAR(255) NOT NULL DEFAULT "	密码
salt	CHAR(32) NOT NULL DEFAULT "	密码盐
email	VARCHAR(128) NOT NULL DEFAULT "	邮箱
mobile	CHAR(11) NOT NULL DEFAULT "	手机号
level	TINYINT UNSIGNED NOT NULL DEFAULT 0	用户级别
money	DECIMAL(10，2) UNSIGNED NOT NULL DEFAULT 0	金额
gender	TINYINT UNSIGNED NOT NULL DEFAULT 0	性别
qq	VARCHAR(20) NOT NULL DEFAULT "	QQ
is_active	TINYINT UNSIGNED NOT NULL DEFAULT 0	是否激活
reg_time	DATETIME DEFAULT NULL	注册时间
create_time	DATETIME NO TNULL DEFAULT CURRENT_TIMESTAMP	创建时间
update_time	DATETIME DEFAULT NULL	更新时间

下面根据表 4-23 所示的表结构，完成用户表的创建，对应的 SQL 语句如下。

```
CREATE TABLE sh_user (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '用户 id',
  name VARCHAR(100) NOT NULL UNIQUE DEFAULT '' COMMENT '用户名',
  password VARCHAR(255) NOT NULL DEFAULT '' COMMENT '密码',
  salt CHAR(32) NOT NULL DEFAULT '' COMMENT '密码盐',
  email VARCHAR(128) NOT NULL DEFAULT '' COMMENT '邮箱',
  mobile CHAR(11) NOT NULL DEFAULT '' COMMENT '手机号',
  level TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '用户级别',
  money DECIMAL(10, 2) UNSIGNED NOT NULL DEFAULT 0 COMMENT '金额',
  gender TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '性别',
  qq VARCHAR(20) NOT NULL DEFAULT '' COMMENT 'QQ',
  is_active TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '是否激活',
```

```
  reg_time DATETIME DEFAULT NULL COMMENT '注册时间',
  create_time DATETIME NOT DULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  update_time DATETIME DEFAULT NULL COMMENT '更新时间'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

需要注意的是，考虑到 Web 项目的安全性，应对用户的密码进行加密，不能明文存储，否则一旦发生数据泄露，将会给网站带来极大的损失。密码的加密方式有很多种，比较常见的是 MD5 加盐算法，下面通过具体 SQL 语句进行演示。

```
mysql> SELECT MD5(CONCAT(MD5('password'), 'salt'));
+--------------------------------------+
| MD5(CONCAT(MD5('password'), 'salt')) |
+--------------------------------------+
| d514dee5e76bbb718084294c835f312c     |
+--------------------------------------+
1 row in set (0.00 sec)
```

在上述 SQL 语句中，MD5('password')表示使用 MD5()函数对 password 字符串进行 MD5(消息摘要算法第五版)运算，运算的结果是一个字符串，由 32 个字符构成。CONCAT()函数用于字符串拼接，此处是在 MD5('password')运算获得的字符串基础上拼接了 salt 字符串。最后，再次使用 MD5()函数对拼接结果进行一次 MD5 运算。

MD5 算法具有不可逆性，通过计算后的结果将无法还原成原文，而对于完全相同的内容，其计算后的结果也是相同的，因此 MD5 算法经常用于密码的存储。但这种方式也有缺点，如果将世界上所有的密码与计算结果全部存储起来进行检索，则密码就会被逆向查找出来。为了解决这个问题，通常会为每个用户生成一个 salt(盐)，用于在对密码进行 MD5 计算时加入 salt，以提高密码加密的复杂度，使密码更不容易被破解。

用户表创建完成后，为用户表添加测试数据，具体 SQL 语句如下。

```
INSERT INTO sh_user (id, name, password, salt, money, is_active) VALUES
(1, 'Alex', MD5(CONCAT(MD5('123'), 'salt1')), 'salt1', 1000, 1),
(2, 'Bill', MD5(CONCAT(MD5('123'), 'salt2')), 'salt2', 1000, 1);
```

在上述测试数据中，password 字段对密码进行了加密，其中 123 是密码原文，Alex 和 Bill 的密码相同，但 salt 不同，所以密码的加密结果也不同。在实际开发中，读者可以借助一些函数生成一个随机字符串，作为 salt 使用，如 MD5(RAND())。

接下来查看 sh_user 表中的记录，观察密码加密结果，如下所示。

```
mysql> SELECT id, name, password, salt FROM sh_user;
+----+------+----------------------------------+-------+
| id | name | password                         | salt  |
+----+------+----------------------------------+-------+
|  1 | Alex | a3a5b6c3c00b37a4c53f82d1ce8bb86c | salt1 |
|  2 | Bill | 8d890cf2ffd7af2c367537c0d6ae99df | salt2 |
+----+------+----------------------------------+-------+
2 rows in set (0.00 sec)
```

4.4.8 评论表

当用户购买商品后，可以发表评论。对于已经发表的评论，可以进行追加。商品评论表的具体结构如表 4-24 所示。

表 4-24 商品评论表(sh_goods_comment)

字 段 名	数据类型和约束	说 明
id	INT UNSIGNED PRIMARY KEY AUTO_INCREMENT	评论 id
parent_id	INT UNSIGNED NOT NULL DEFAULT 0	上级评论 id
user_id	INT UNSIGNED NOT NULL DEFAULT 0	用户 id
goods_id	INT UNSIGNED NOT NULL DEFAULT 0	商品 id
content	TEXT NOT NULL	评论内容
is_staff	TINYINT UNSIGNED NOT NULL DEFAULT 0	工作人员
is_show	TINYINT UNSIGNED NOT NULL DEFAULT 0	是否显示
is_del	TINYINT UNSIGNED NOT NULL DEFAULT 0	是否删除
create_time	DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP	创建时间
update_time	DATETIME DEFAULT NULL	更新时间

在表 4-24 中，is_staff 字段表示该评论是否为网站的后台用户(即工作人员)发表的，用于工作人员回复用户的评论、解答疑问等。

下面根据表 4-24 的结构，创建商品评论表，对应的 SQL 语句如下。

```
CREATE TABLE sh_goods_comment (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '评论 id',
  parent_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '上级评论 id',
  user_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '用户 id',
  goods_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '商品 id',
  content TEXT NOT NULL COMMENT '评论内容',
  is_staff TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '是否为工作人员',
  is_show TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '是否显示',
  is_del TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '是否删除',
  create_time DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  update_time DATETIME DEFAULT NULL COMMENT '更新时间'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

数据表创建完成后，为商品评论表添加测试数据，具体 SQL 语句如下。

```
INSERT INTO `sh_goods_comment` (id, user_id, goods_id, content,
is_show, create_time) VALUES
(1, 1, 8, '好', 0, '2017-11-08 00:00:00'),
(2, 2, 10, '不错', 1, '2017-12-03 00:00:00'),
(3, 3, 9, '满意', 1, '2017-12-30 00:00:00'),
(4, 4, 4, '携带方便', 1, '2018-01-19 00:00:00'),
```

```
(5, 4, 7, '中低音效果特别棒', 1, '2018-01-19 00:00:00'),
(6, 5, 8, '卡机', 1, '2018-01-22 00:00:00'),
(7, 6, 5, '黑夜拍照也清晰', 1, '2018-02-15 00:00:00'),
(8, 7, 9, '掉色、有线头', 0, '2018-03-03 00:00:00'),
(9, 4, 9, '还行', 1, '2018-04-05 00:00:00'),
(10, 8, 9, '特别彰显气质', 1,'2018-04-16 00:00:00');
```

4.5　动手实践：商品购物流程设计

数据库的学习在于多看、多学、多想、多动手，只有将理论与实际相结合，才能够体现出数据库开发与管理的重要性，展现知识学习的价值与力量。接下来请结合本章所学的知识完成电子商务网站中的商品购物流程的数据库设计。

【实践目标】

此实践的目标是能够根据文字提示，完成商品购物流程的数据库设计。

【实践需求】

用户在电子商务网站中购买商品时，用户将想要购买的商品添加到购物车，填写收货地址，然后下订单，等待收货。在收到货以后，可以对商品进行打分评价。请动手实现购物车、收货地址、订单、商品评分的数据表设计。

【手动实践】

1. 购物车

用户可以将想要购买的商品添加到购物车，从而方便一次购买多件商品。将商品添加到购物车后不会影响商品的库存。接下来通过表 4-25 展示购物车表的结构。

表 4-25　购物车表（sh_user_shopcart）

字　段　名	数据类型和约束	说　　明
id	INT UNSIGNED PRIMARY KEY AUTO_INCREMENT	购物车 id
user_id	INT UNSIGNED NOT NULL DEFAULT 0	用户 id
goods_id	INT UNSIGNED NOT NULL DEFAULT 0	商品 id
goods_price	DECIMAL(10，2) UNSIGNED NOT NULL DEFAULT 0	单价
goods_num	INT UNSIGNED NOT NULL DEFAULT 0	购买件数
is_select	TINYINT UNSIGNED NOT NULL DEFAULT 0	是否选中
create_time	DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP	创建时间
update_time	DATETIME DEFAULT NULL	更新时间

在表 4-25 中，goods_price 字段表示商品加入到购物车时的单价，并不一定是商品的最新价格，这是因为商品的价格有可能会变化，当变化时，购物车会显示商品的最新价格，并根

据 goods_price 保存的价格计算该商品的价格浮动变化，以提醒用户该商品已经涨价或降价。

下面根据表 4-25 的结构，创建购物车表，对应的 SQL 语句如下。

```
CREATE TABLE sh_user_shopcart (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '购物车 id',
  user_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '用户 id',
  goods_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '商品 id',
  goods_price DECIMAL(10, 2) UNSIGNED NOT NULL DEFAULT 0 COMMENT '单价',
  goods_num INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '购买件数',
  is_select TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '是否选中',
  create_time DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  update_time DATETIME DEFAULT NULL COMMENT '更新时间'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

2. 收货地址

当用户决定下订单前，需要选择收货地址。一个用户可以有多个收货地址，并根据实际情况选择某一个作为默认地址。接下来通过表 4-26 展示收货地址表的结构。

表 4-26　收货地址表(sh_user_address)

字　段　名	数据类型和约束	说　　明
id	INT UNSIGNED PRIMARY KEY AUTO_INCREMENT	地址 id
user_id	INT UNSIGNED NOT NULL DEFAULT 0	用户 id
is_default	TINYINT UNSIGNED NOT NULL DEFAULT 0	是否默认
province	VARCHAR(20) NOT NULL DEFAULT ''	省
city	VARCHAR(20) NOT NULL DEFAULT ''	市
district	VARCHAR(20) NOT NULL DEFAULT ''	区
address	VARCHAR(255) NOT NULL DEFAULT ''	具体地址
zip	VARCHAR(20) NOT NULL DEFAULT ''	邮编
consignee	VARCHAR(20) NOT NULL DEFAULT ''	收件人
phone	VARCHAR(20) NOT NULL DEFAULT ''	联系电话
create_time	DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP	创建时间
update_time	DATETIME DEFAULT NULL	更新时间

下面根据表 4-26 的结构，创建收货地址表，对应的 SQL 语句如下。

```
CREATE TABLE sh_user_address (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '地址 id',
  user_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '用户 id',
  is_default TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '是否默认',
  province VARCHAR(20) NOT NULL DEFAULT '' COMMENT '省',
```

```
    city VARCHAR(20) NOT NULL DEFAULT '' COMMENT '市',
    district VARCHAR(20) NOT NULL DEFAULT '' COMMENT '区',
    address VARCHAR(255) NOT NULL DEFAULT '' COMMENT '具体地址',
    zip VARCHAR(20) NOT NULL DEFAULT '' COMMENT '邮编',
    consignee VARCHAR(20) NOT NULL DEFAULT '' COMMENT '收件人',
    phone VARCHAR(20) NOT NULL DEFAULT '' COMMENT '联系电话',
    create_time DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
    update_time DATETIME DEFAULT NULL COMMENT '更新时间'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

3. 订单

当用户确定购买一件或多件商品后，就可以下订单并进行支付。订单表的具体结构如表 4-27 所示。

表 4-27 订单表(sh_order)

字 段 名	数据类型和约束	说 明
id	INT UNSIGNED PRIMARY KEY AUTO_INCREMENT	订单 id
user_id	INT UNSIGNED NOT NULL DEFAULT 0	用户 id
total_price	DECIMAL(10，2) UNSIGNED NOT NULL DEFAULT 0	订单总价
order_price	DECIMAL(10，2) UNSIGNED NOT NULL DEFAULT 0	应付金额
province	VARCHAR(20) NOT NULL DEFAULT ''	省
city	VARCHAR(20) NOT NULL DEFAULT ''	市
district	VARCHAR(20) NOT NULL DEFAULT ''	区
address	VARCHAR(255) NOT NULL DEFAULT ''	具体地址
zip	VARCHAR(20) NOT NULL DEFAULT ''	邮编
consignee	VARCHAR(20) NOT NULL DEFAULT ''	收件人
phone	VARCHAR(20) NOT NULL DEFAULT ''	联系电话
is_valid	TINYINT UNSIGNED NOT NULL DEFAULT 0	是否有效
is_cancel	TINYINT UNSIGNED NOT NULL DEFAULT 0	是否取消
is_pay	TINYINT UNSIGNED NOT NULL DEFAULT 0	是否付款
status	TINYINT UNSIGNED NOT NULL DEFAULT 0	物流状态
is_del	TINYINT UNSIGNED NOT NULL DEFAULT 0	是否删除
create_time	DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP	创建时间
update_time	DATETIME DEFAULT NULL	更新时间

在表 4-27 中，“是否有效”用于在订单创建后由工作人员进行确认；“是否取消”用于取消订单；“是否付款”表示用户是否付款；“物流状态”的值有 3 种，0 表示未发货，1 表示已发货，2 表示用户确认收货；“是否删除”值为 0 表示不删除，1 表示删除到回收站，2 表示从回

收站中删除。每当订单的状态(包括订单创建、工作人员确认、取消订单、付款、发货、确认收货)发生变化时,就需要记录日志,日志中保存操作类型、时间、日志信息,以及用户或工作人员的留言(如取消订单的理由)等信息。

由于一个订单可以包含多件商品,因此还需要通过订单商品表保存订单中的商品,具体结构如表 4-28 所示。

表 4-28 订单商品表(sh_order_goods)

字 段 名	数据类型和约束	说 明
id	INT UNSIGNED PRIMARY KEY AUTO_INCREMENT	id
order_id	INT UNSIGNED NOT NULL DEFAULT 0	订单 id
goods_id	INT UNSIGNED NOT NULL DEFAULT 0	商品 id
goods_name	VARCHAR(120) NOT NULL DEFAULT ''	商品名称
goods_num	INT UNSIGNED NOT NULL DEFAULT 0	购买数量
goods_price	DECIMAL(10, 2) UNSIGNED NOT NULL DEFAULT 0	商品单价
user_note	VARCHAR(255) NOT NULL DEFAULT ''	用户备注
staff_note	VARCHAR(255) NOT NULL DEFAULT ''	卖家备注

下面根据表 4-27 和表 4-28 的结构,创建订单表和订单商品表,对应的 SQL 语句如下。

```
#订单表
CREATE TABLE sh_order (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '订单 id',
  user_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '用户 id',
  total_price DECIMAL(10, 2) UNSIGNED NOT NULL DEFAULT 0 COMMENT '订单总价',
  order_price DECIMAL(10, 2) UNSIGNED NOT NULL DEFAULT 0 COMMENT '应付金额',
  province VARCHAR(20) NOT NULL DEFAULT '' COMMENT '省',
  city VARCHAR(20) NOT NULL DEFAULT '' COMMENT '市',
  district VARCHAR(20) NOT NULL DEFAULT '' COMMENT '区',
  address VARCHAR(255) NOT NULL DEFAULT '' COMMENT '具体地址',
  zip VARCHAR(20) NOT NULL DEFAULT '' COMMENT '邮编',
  consignee VARCHAR(20) NOT NULL DEFAULT '' COMMENT '收件人',
  phone VARCHAR(20) NOT NULL DEFAULT '' COMMENT '联系电话',
  is_valid TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '是否有效',
  is_cancel TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '是否取消',
  is_pay TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '是否付款',
  status TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '物流状态',
  is_del TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '是否删除',
  create_time DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  update_time DATETIME DEFAULT NULL COMMENT '更新时间'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
#订单商品表
CREATE TABLE sh_order_goods (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT 'id',
  order_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '订单 id',
```

```
  goods_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '商品 id',
  goods_name VARCHAR(120) NOT NULL DEFAULT '' COMMENT '商品名称',
  goods_num INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '购买数量',
  goods_price DECIMAL(10, 2) UNSIGNED NOT NULL DEFAULT 0 COMMENT '单价',
  user_note VARCHAR(255) NOT NULL DEFAULT '' COMMENT '用户备注',
  staff_note VARCHAR(255) NOT NULL DEFAULT '' COMMENT '卖家备注'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

4．商品评分

当用户购买商品后，可以对商品进行评分，共有 1～5 分可选。商品评分表的具体结构如表 4-29 所示。

表 4-29　商品评分表(sh_goods_score)

字　段　名	数据类型和约束	说　　明
id	INT UNSIGNED PRIMARY KEY AUTO_INCREMENT	评分 id
user_id	INT UNSIGNED NOT NULL DEFAULT 0	用户 id
goods_id	INT UNSIGNED NOT NULL DEFAULT 0	商品 id
goods_score	TINYINT UNSIGNED NOT NULL DEFAULT 0	商品评分
service_score	TINYINT UNSIGNED NOT NULL DEFAULT 0	服务评分
express_score	TINYINT UNSIGNED NOT NULL DEFAULT 0	物流评分
is_invalid	TINYINT UNSIGNED NOT NULL DEFAULT 0	是否无效
create_time	DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP	评分时间

在表 4-29 中，“是否无效”用于商家遇到恶意差评时，工作人员可以将评分设为无效，无效的评分将不会参与到商品的平均评分计算中。

下面根据表 4-29 的结构，创建商品评分表，对应的 SQL 语句如下。

```
CREATE TABLE sh_goods_score (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '评分 id',
  user_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '用户 id',
  goods_id INT UNSIGNED NOT NULL DEFAULT 0 COMMENT '商品 id',
  goods_score TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '商品评分',
  service_score TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '服务评分',
  express_score TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '物流评分',
  is_invalid TINYINT UNSIGNED NOT NULL DEFAULT 0 COMMENT '是否无效',
  create_time DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '评分时间'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

4.6 本章小结

本章主要讲解了数据库设计的基本理论和具体实战，读者应掌握数据库设计的基本流程、数据建模工具的使用，并深入理解数据库范式的作用和局限。通过本章的学习，读者应具备分析用户需求，设计合理、规范的数据库的能力。

4.7 课后练习

一、填空题

1. 在 E-R 图中，实体使用________图形来表示。
2. 实体所具有的某一特征称为实体的________。
3. 将 E-R 图转换到关系模式时，实体与联系都可以表示成________。
4. 设计数据库的存储结构属于数据库设计的________阶段。
5. 在 E-R 图中的联系可以与________个实体有关。

二、判断题

1. 数据的逻辑结构具体反映数据在计算机中的存储方式。(　　)
2. 数据库正式投入运行标志着数据库运行和维护工作的开始。(　　)
3. 在关系模型中，同一表中的不同属性命名可以相同。(　　)
4. 消除了部分函数依赖的 1NF 的关系模式，必定是 2NF。(　　)
5. 任何由两个属性组成的关系不可能是 3NF。(　　)

三、选择题

1. 绘制 E-R 图的 3 个基本要素是(　　)。
   A. 实体、属性、关键字　　B. 属性、实体、联系
   C. 属性、数据类型、实体　　D. 约束、属性、实体
2. 绘制 E-R 图属于数据库设计的(　　)阶段。
   A. 需求分析　　B. 概念数据库设计
   C. 逻辑数据库设计　　D. 物理数据库设计
3. 将 E-R 图转换为数据模型属于数据库设计的(　　)阶段。
   A. 数据库实施　　B. 概念数据库设计
   C. 逻辑数据库设计　　D. 物理数据库设计
4. 数据冗余可能会引起的问题有(　　)。
   A. 读取异常　　B. 更新异常　　C. 插入异常　　D. 删除异常
5. 下列关于数据库范式说法正确的是(　　)。
   A. 1NF 遵从原子性和唯一性，且字段不可再分

B. 2NF 要求非主键字段需要依赖主键
C. 3NF 要求非主键字段不能相互依赖
D. 各个范式之间互不依赖，只需满足当前范式的要求即可

四、简答题

1. 请简述数据库设计规范化的必要性。
2. 请分析数据库范式 1NF、2NF、3NF 的区别。

五、实训题

1. 请完成电子商务网站用户等级的数据表设计。
2. 请完成电子商务网站用户订阅、用户收藏的数据表设计。

第 5 章 单表操作

学习目标

- 掌握复制表结构与数据的操作
- 掌握数据的排序、限量与分组
- 掌握常用聚合函数与比较函数
- 掌握 MySQL 常用运算符的使用

在前面的章节中已经学习了数据表的创建、数据类型、约束、字符集的设置，以及数据的基本增、删、改、查操作。但实际需求会更加复杂，前面学习过的内容不能够完全满足开发需求，所以需要深入学习更多的数据操作。例如，为数据表插入大量的测试数据，对查询的数据进行筛选、分组、排序或限量。本章将围绕数据库中的单表操作进行详细讲解。

5.1 数据操作

5.1.1 复制表结构和数据

1. 复制已有的表结构

在开发时，若需要创建一个与已有数据表相同结构的数据表时，可以通过以下的语法完成表结构的复制。基本语法格式如下。

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS]表名
{ LIKE 旧表名 | (LIKE 旧表名) }
```

在上述语法中，仅能从“旧表名”中复制一份相同的表结构，但不会复制表中保存的数据。其中，“{}”表示语法在使用时可以任选其中一种，“|”表示或的意思。因此，在复制已有的表结构时，可以使用“LIKE 旧表名”或“(LIKE 旧表名)”中的任意一种语法格式。

下面从第 4 章创建的 shop 数据库中，复制一份与 sh_goods 数据表相同结构的 my_goods 表到 mydb 数据库中，具体 SQL 语句如下。

```
mysql> USE shop;
Database changed
mysql> CREATE  TABLE  mydb.my_goods LIKE sh_goods;
Query OK, 0 rows affected (0.07 sec)
```

按以上步骤创建完成后，可以利用 SHOW CREATE TABLE 查看 my_goods 表的结构，具体 SQL 语句如下。

```
mysql> SHOW CREATE TABLE mydb.my_goods\G
* * * * * * * * * * * * * * 1. row * * * * * * * * * * * * * *
       Table: my_goods
Create Table: CREATE TABLE `my_goods` (
   `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '商品 id',
  `category_id` int(10) unsigned NOT NULL DEFAULT '0' COMMENT '分类 id',
   ：此处省略部分字段
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

从上述结果可知，只需一行操作就可以依据已有的表创建出与其相同结构的表。

2. 复制已有的表数据

数据复制也可称为蠕虫复制，是新增数据的一种方式，它是从已有的数据中获取数据，并且将获取到的数据插入到对应的数据表中，实现成倍的增加。需要注意的是，此种方式获取数据与插入数据的表结构要相同，否则可能会遇到插入不成功的情况。基本语法格式如下。

```
INSERT [INTO] 数据表名 1 [(字段列表)] SELECT [(字段列表)] FROM 数据表名 2;
```

在上述语法中，数据表名 1 和数据表名 2 通常使用的是同一个表（如 my_goods 表），从而可在短期内快速增加表的数据量，测试表的压力以及效率等，相关内容会在本书数据库优化部分讲解，此处读者了解即可。

下面利用以上语法从 sh_goods 表中复制数据到 my_goods 表中，具体 SQL 语句如下。

```
mysql> INSERT INTO mydb.my_goods SELECT * FROM sh_goods;
Query OK, 10 rows affected (0.01 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

执行完上述 SQL 语句后，使用 SELECT 查看商品数据的添加情况，会看到数据已从 sh_goods 表完全复制到了 my_goods 表中，这里不再演示。

需要注意的是，若数据表中含有主键，而主键又具有唯一性，所以在数据复制时还要考虑主键冲突的问题。例如，通过以下方式再向 my_goods 表中添加数据，系统会报“主键重复”的错误。具体 SQL 语句如下。

```
mysql> INSERT INTO mydb.my_goods SELECT * FROM sh_goods;
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
```

对于以上主键冲突的问题，数据复制时可以指定除 id 主键外的任意字段完成。具体 SQL 语句如下。

```
mysql> INSERT INTO mydb.my_goods (category_id, name, keyword, price,
    -> content) SELECT category_id, name, keyword, price, content
```

```
    ->FROM sh_goods;
Query OK, 10 rows affected (0.00 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

多学一招：临时表的使用

临时表指的是一种仅在当前会话中可见，并在当前会话关闭时自动删除的数据表。它主要用于临时存储数据。临时表的语法很简单，只需在 CREATE 与 TABLE 关键字中间添加 TEMPORARY 即可。示例如下。

```
#方式 1：创建临时表
CREATE TEMPORARY TABLE mydb.tmp_table1 (id int);
#方式 2：创建临时表
CREATE TEMPORARY TABLE mydb.tmp_table2 SELECT id, name FROM shop.sh_goods;
```

在上述语句中，创建临时表时指定的数据库可以是 MySQL 服务器中存在的数据库，也可以是不存在的数据库。若数据库不存在，操作临时表时必须使用“数据库.临时表名”指定临时表所在的数据库。

除此之外，临时表中数据的操作与普通表相同，都可以进行 SELECT、INSERT、UPDATE 和 DELETE 操作。这里不再演示。

需要注意的是，SHOW TABLES 不能查看指定数据库下有哪些临时表，并且临时表的表名必须使用 ALTER TABLE 修改，而不能使用 RENAME TABLE…TO 修改。

5.1.2 解决主键冲突

在对数据表插入数据时，若表中的主键含有实际的业务意义，因此在插入数据时若不能确定对应的主键是否存在，往往会出现主键冲突的情况。例如，mydb.my_goods 表经过数据复制以后，再插入编号为 20 的商品信息（橡皮，文具类，用于修正书写错误），具体 SQL 语句及执行结果如下。

```
mysql> INSERT INTO mydb.my_goods (id, name, content, keyword)
    -> VALUES (20, '橡皮', '修正书写错误', '文具');
ERROR 1062 (23000): Duplicate entry '20' for key 'PRIMARY'
```

从上述的执行结果可知，系统提示插入数据的主键发生冲突。若要解决这类问题，MySQL 中提供了两种方式，分别为主键冲突更新和主键冲突替换。

1. 主键冲突更新

主键冲突更新操作指的是，当插入数据的过程中若发生主键冲突，则插入数据操作利用更新的方式实现。基于语法格式如下。

```
INSERT [INTO] 数据表名 [(字段列表)] {VALUES | VALUE} (值列表)
ON DUPLICATE KEY UPDATE 字段名 1=新值 1[,字段名 2=新值 2] …;
```

从上述语法可知，在 INSERT 语句后添加 ON DUPLICATE KEY UPDATE 可在发生

主键冲突时，更新此条记录中通过"字段名 1 = 新值 1[，字段名 2 = 新值 2] …"设置的字段名对应的新值。

例如，修改以上发生主键冲突的插入语句，具体 SQL 语句及执行结果如下。

```
mysql> INSERT INTO mydb.my_goods (id, name, content, keyword)
    -> VALUES (20, '橡皮', '修正书写错误', '文具')
    -> ON DUPLICATE KEY UPDATE name = '橡皮', content = '修正书写错误',
    -> keyword = '文具';
Query OK, 2 rows affected (0.00 sec)
mysql> SELECT name, content, keyword FROM mydb.my_goods WHERE id = 20;
+------+------------+---------+
| name | content    | keyword |
+------+------------+---------+
|橡皮  | 修正书写错误 | 文具    |
+------+------------+---------+
1 row in set (0.00 sec)
```

以上执行结果中，当插入的记录与数据表中已存在的记录主键冲突时，返回的结果为"2 rows affected"。

2. 主键冲突替换

主键冲突替换操作指的是，当插入数据的过程中若发生主键冲突，则删除此条记录，并重新插入。基于语法格式如下。

```
REPLACE [INTO] 数据表名 [(字段列表)]
{VALUES | VALUE} (值列表) [, (值列表)] …;
```

从上述语法可知，REPLACE 语句与 INSERT 语句的使用类似，区别在于前者每执行一次就会发生两个操作（删除记录和插入记录）。例如，修改以上发生主键冲突的插入语句，具体 SQL 语句及执行结果如下。

```
mysql> REPLACE INTO mydb.my_goods (id, name, content, keyword)
    -> VALUES (20, '橡皮', '修正书写错误', '文具');
Query OK, 2 rows affected (0.00 sec)
mysql> SELECT name, content, keyword FROM mydb.my_goods WHERE id = 20;
+------+------------+----------+
| name | content    | keyword  |
+------+------------+----------+
|橡皮  | 修正书写错误 | 文具     |
+------+------------+----------+
1 row in set (0.00 sec)
```

从以上的执行结果可知，REPLACE 替换与 ON DUPLICATE KEY UPDATE 更新都能解决插入数据时主键冲突的问题，但 REPLACE 更适合插入数据字段特别多的情况。

5.1.3　清空数据

除了第 2 章讲解的 DELETE 语句可以删除数据外，在 MySQL 中还可以利用

TRUNCATE 清空指定数据表中的全部数据。其基本语法格式如下。

```
TRUNCATE [TABLE] 表名
```

下面演示清空 mydb 数据库下的 my_goods 表，具体 SQL 语句及执行结果如下。

```
mysql> TRUNCATE TABLE mydb.my_goods;
Query OK, 0 rows affected (0.08 sec)
```

需要注意的是，TRUNCATE 操作虽然与 DELETE 语句的使用非常相似，但是两者在本质上有一定的区别，具体如下。

- 实现方式不同：TRUNCATE 本质上先执行删除(DROP)数据表的操作，然后再根据有效的表结构文件(.frm)重新创建数据表的方式来实现数据清空操作。而 DELETE 语句则是逐条地删除数据表中保存的记录。
- 执行效率不同：在针对大型数据表(如千万级的数据记录)时，TRUNCATE 清空数据的实现方式决定了它比 DELETE 语句删除数据的方式执行效率更高。
- 对 AUTO_INCREMENT 的字段影响不同：TRUNCATE 清空数据后，再次向表中添加数据，自动增长字段会从默认的初始值重新开始，而使用 DELETE 语句删除表中的记录时，则不影响自动增长值。
- 删除数据的范围不同：TRUNCATE 语句只能用于清空表中的所有记录，而 DELETE 语句可通过 WHERE 指定删除满足条件的部分记录。
- 返回值含义不同：TRUNCATE 操作的返回值一般是无意义的，而 DELETE 语句则会返回符合条件被删除的记录数。
- 所属 SQL 语言的不同组成部分：DELETE 语句属于 DML 数据操作语句，而 TRUNCATE 通常被认为是 DDL 数据定义语句。

为了读者更好地理解，重新使用 5.1.1 小节中数据复制的方式完成 my_goods 表的数据添加，对比 TRUNCATE 与 DELETE 数据表 my_goods 后，重新插入一条记录的效果。

(1) 为 my_goods 插入 10 条记录后，使用 TRUNCATE 清空并重新插入数据。

```
mysql> TRUNCATE TABLE mydb.my_goods;
Query OK, 0 rows affected (0.01 sec)
mysql> INSERT INTO mydb.my_goods (name, content, keyword)
    -> VALUES ('苹果', '一种很有营养的水果', '水果');
Query OK, 1 row affected (0.00 sec)
mysql> SELECT id, name, content, keyword FROM mydb.my_goods;
+----+------+-------------------+------------+
| id | name | content           | keyword    |
+----+------+-------------------+------------+
| 1  | 苹果 | 一种很有营养的水果 | 水果       |
+----+------+-------------------+------------+
1 row in set (0.00 sec)
```

(2) 为 my_goods 插入 10 条记录后，使用 DELETE 删除并重新插入数据。

```
mysql> DELETE FROM mydb.my_goods;
Query OK, 10 rows affected (0.00 sec)
mysql> INSERT INTO mydb.my_goods (name, content, keyword)
    -> VALUES ('苹果', '一种很有营养的水果', '水果');
Query OK, 1 row affected (0.00 sec)
mysql> SELECT id, name, content, keyword FROM mydb.my_goods;
+----+------+--------------------+------------+
| id | name | content            | keyword    |
+----+------+--------------------+------------+
| 11 | 苹果 | 一种很有营养的水果 | 水果       |
+----+------+--------------------+------------+
1 row in set (0.00 sec)
```

从上述的操作中不难看出 TRUNCATE 操作和 DELETE 语句在删除数据的区别，具体如表 5-1 所示。

表 5-1　TRUNCATE 与 DELETE 对比

操　作	返　回　值	id(插入字段增长值)	执行效率
TRUNCATE	0 rows affected	1	0.01s
DELETE	10 rows affected	11	0.00s

在表 5-1 中，DELETE 的返回值表示有 10 条记录受影响，而 TRUNCATE 的返回值为 0，明显无实际意义；删除数据后，再次新增一条记录后，查询到的商品 id 值明显不同，TRUNCATE 后 id 字段从默认值 1 开始增长，而 DELETE 后 id 值则继续从 10 开始增长，因此最后结果为 11。

需要注意的是，当删除的数据量很小时，DELETE 的执行效率要比 TRUNCATE 高；只有删除的数据量很大时，才能看出 TRUNCATE 的执行效率比 DELETE 高。例如，my_goods 表含有 20480 条记录时，使用 TRUNCATE 清空数据的执行时间仍然为 0.01 秒，而 DELETE 语句的执行时间会增加到 0.17 秒左右。因此，在实际开发时具体使用哪种方式执行删除操作，需要根据实际需求进行合理的选择。

5.1.4　去除重复记录

实际应用中，出于对数据的分析需求，有时需要去除查询记录中重复的数据。例如，查看商品表中共有几种分类。此时可以使用 SELECT 语句的选项，其基本语法格式如下。

```
SELECT select 选项 字段列表 FROM 数据表
```

在上述语法中，“select 选项”默认值为 All，表示保存所有查询到的记录；当设置为 DISTINCT 时，表示去除重复记录，只保留一条。需要注意的是，当查询记录的字段有多个时，必须所有字段的值完全相同才被认为是重复记录。

为了方便案例的演示，本章以下所有小节均使用 shop 数据库下的 sh_goods 表进行操作。下面查看 sh_goods 表中所有的 keyword 字段的值，具体 SQL 语句如下。

```
mysql> SELECT keyword FROM sh_goods;
+-----------+
| keyword   |
+-----------+
| 文具      |
| 文具      |
| 文具      |
| 电子产品  |
| 电子产品  |
| 电子产品  |
| 电子产品  |
| 电子产品  |
| 服装      |
| 服装      |
+-----------+
10 rows in set (0.00 sec)
```

从上述执行结果可知，查询出的 keyword 字段值有 3 条为“文具”，5 条为“电子产品”，2 条为“服装”，即使有重复的数据，默认情况下依然保存了所有查询到的记录。

接下来，查看 sh_goods 表中去除重复记录的 keyword 字段值，具体 SQL 语句如下。

```
mysql> SELECT DISTINCT keyword FROM sh_goods;
+------------+
| keyword    |
+------------+
| 文具       |
| 电子产品   |
| 服装       |
+------------+
3 rows in set (0.00 sec)
```

从上述执行结果可知，此次的查询结果仅有 3 条记录，分别为文具、电子产品和服装，不再包含重复的记录。

5.2 排序与限量

在电子商务网站迅速发展的今天，商品的种类与数量数以万计，甚至更多。因此，人们在查看某种商品时经常需要对其进行排序，让满足要求的数据显示到最前面，方便进一步操作。同时，为了提高执行的效率，经常需要对操作的数据进行限量。例如，仅查看符合要求的 10 条记录。本节将针对 MySQL 中排序和限量操作进行详细讲解。

5.2.1 排序

在项目开发时，为了使查询的数据结果满足用户的要求，通常会对查询出的数据进行上升或下降的排序。MySQL 针对不同的开发需求提供两种排序的方式，分别为单字段排序和多字段排序，接下来将对这两种排序方式的语法及使用进行详细讲解。

1. 单字段排序

单字段排序指的是查询时仅按照一个指定字段进行升序或降序排序。其基本语法格式如下。

```
SELECT  * |{字段列表} FROM 数据表名
ORDER BY 字段名 [ASC | DESC];
```

在上述语法中,ASC 表示升序,DESC 表示降序。而 ORDER BY 默认值为 ASC。

下面按照商品价格从高到低依次显示 sh_goods 表中的所有商品。具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, name, price FROM sh_goods ORDER BY price DESC;
+----+-----------+---------+
| id | name      | price   |
+----+-----------+---------+
|  4 | 超薄笔记本 | 5999.00 |
|  8 | 办公电脑   | 2000.00 |
|  5 | 智能手机   | 1999.00 |
|  9 | 收腰风衣   |  299.00 |
|  7 | 头戴耳机   |  109.00 |
|  6 | 桌面音箱   |   69.00 |
| 10 | 薄毛衣     |   48.00 |
|  2 | 钢笔       |   15.00 |
|  3 | 碳素笔     |    1.00 |
|  1 | 2B 铅笔    |    0.50 |
+----+-----------+---------+
10 rows in set (0.00 sec)
```

从以上的执行结果可知,查询到的商品显示顺序以价格为标准,从高到低依次显示。

2. 多字段排序

当在开发中需要根据多个条件对查询的数据进行排序时,可以采用多字段排序。其基本语法格式如下。

```
SELECT  * |{字段列表} FROM 数据表名
ORDER BY 字段名 1 [ASC | DESC] [, 字段名 2 [ASC | DESC]]…;
```

在上述语法中,多字段排序首先按照字段名 1 进行排序,当字段 1 的值相同时,再按照字段名 2 进行排序,依此类推。

下面查询 sh_goods 表中的数据,让数据在显示时首先按商品分类 category_id 升序排序,然后再按商品价格 price 从高到低依次排序。具体 SQL 语句及执行结果如下。

```
mysql> SELECT category_id, id, name, price FROM sh_goods
    -> ORDER BY category_id, price DESC;
+-------------+----+-----------+---------+
| category_id | id | name      | price   |
```

```
+-------------+----+------------+---------+
|           3 |  2 | 钢笔       |   15.00 |
|           3 |  3 | 碳素笔     |    1.00 |
|           3 |  1 | 2B 铅笔    |    0.50 |
|           6 |  5 | 智能手机   | 1999.00 |
|           8 |  6 | 桌面音箱   |   69.00 |
|           9 |  7 | 头戴耳机   |  109.00 |
|          10 |  8 | 办公电脑   | 2000.00 |
|          12 |  4 | 超薄笔记本 | 5999.00 |
|          15 |  9 | 收腰风衣   |  299.00 |
|          16 | 10 | 薄毛衣     |   48.00 |
+-------------+----+------------+---------+
10 rows in set (0.00 sec)
```

在上述执行结果中，查询的所有数据首先按 category_id 升序排序，相同 category_id 值的记录按照 price 字段降序排序。

此外由于数据表的字符集是 utf8，当排序的字段为中文时，默认不会按照中文拼音的顺序排序。那么在不改变数据表结构的情况下，可以使用"CONVERT(字段名 USING gbk)"函数强制让指定的字段按中文排序。

值得一提的是，在按照指定字段进行升序排列时，如果某条记录的字段值为 NULL，则系统会将 NULL 看作是最小的值，从而将其显示在查询结果中的第一条记录的位置。

5.2.2 限量

对于一次性查询出的大量记录，不仅不便于阅读查看，还会浪费系统效率。为此，MySQL 中提供了一个关键字 LIMIT，可以限定记录的数量，也可以指定查询从哪一条记录开始。其基本语法格式如下。

```
SELECT [select 选项] 字段列表 FROM 数据表名
[WHERE 条件表达式] [ORDER BY 字段 ASC|DESC]
LIMIT [OFFSET,] 记录数;
```

在上述语法中，"记录数"表示限定获取的最大记录数量，也就是说，在"记录数"大于数据表符合要求的实际记录数量时，以实际记录数为准；当 LIMIT 后仅含有此参数时，表示从第 1 条记录开始获取；可选项 OFFSET 表示偏移量，用于设置从哪条记录开始，MySQL 中默认第 1 条记录的偏移量值为 0，第 2 条记录的偏移量值为 1，依此类推。

下面以 sh_goods 表为例，演示限量查询记录与如何获取指定区间的记录。

(1) 限制记录数。查询 sh_goods 表中价格最贵的商品，具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, name, price FROM sh_goods
    -> ORDER BY price DESC LIMIT 1;
+----+------------+---------+
| id | name       | price   |
+----+------------+---------+
|  4 | 超薄笔记本 | 5999.00 |
+----+------------+---------+
1 row in set (0.00 sec)
```

在上述 SQL 语句中，利用商品价格从高到低依次排序，然后再利用 LIMIT 限制查询出的记录数为 1 条，即可获取查询记录中的第 1 条记录，也就是价格最贵的商品。

(2) 获取指定区间的记录。获取指定区间的记录通常在项目开发中用于实现数据的分页展示，从而缓解网络和服务器的压力。例如，从第 1 条记录开始，获取 5 条商品记录，商品记录中包含 id、name 和 price，具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, name, price FROM sh_goods LIMIT 0, 5;
+----+------------+---------+
| id | name       | price   |
+----+------------+---------+
|  1 | 2B铅笔     |    0.50 |
|  2 | 钢笔       |   15.00 |
|  3 | 碳素笔     |    1.00 |
|  4 | 超薄笔记本 | 5999.00 |
|  5 | 智能手机   | 1999.00 |
+----+------------+---------+
5 rows in set (0.00 sec)
```

在上述 SQL 语句中，LIMIT 关键字后的“0”表示第 1 条记录的偏移量，“5”表示从第 1 (偏移量＋1)条记录开始最多获取 5 条记录。

多学一招：排序后限量更新或删除数据

在 MySQL 中除了对查询记录进行排序和限量外，对数据表中记录的更新与删除操作也可以进行排序和限量。其基本语法格式如下。

```
#数据更新的排序与限量
UPDATE 数据表名 SET 字段=新值, … [WHERE 条件表达式]
ORDER BY 字段 ASC|DESC LIMIT 记录数;
#数据删除的排序与限量
DELETE FROM 数据表名 [WHERE 条件表达式]
ORDER BY 字段 ASC|DESC LIMIT 记录数;
```

在上述语法中，UPDATE 和 DELETE 操作中添加 ORDER BY 表示根据指定的字段，按顺序更新或删除符合条件的记录。如果 UPDATE 和 DELETE 操作没有添加 WHERE 条件，则可以使用 LIMIT 来限制更新和删除的数量。

例如，为 sh_goods 表中价格最便宜的两种商品库存设置为 500，具体 SQL 语句及查询结果如下。

```
mysql> UPDATE sh_goods SET stock = 500
    -> ORDER BY price ASC
    -> LIMIT 2;
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0
mysql> SELECT id, name, price, stock FROM sh_goods ORDER BY price;
+----+------------+---------+-------+
| id | name       | price   | stock |
```

```
+----+------------+---------+-------+
| 1  | 2B铅笔      |    0.50 |  500  |
| 3  | 碳素笔      |    1.00 |  500  |
| 2  | 钢笔        |   15.00 |  300  |
| 10 |薄毛衣       |   48.00 |    0  |
| 6  | 桌面音箱    |   69.00 |  750  |
| 7  | 头戴耳机    |  109.00 |    0  |
| 9  | 收腰风衣    |  299.00 |    0  |
| 5  | 智能手机    | 1999.00 |    0  |
| 8  | 办公电脑    | 2000.00 |    0  |
| 4  | 超薄笔记本  | 5999.00 |    0  |
+----+------------+---------+-------+
10 rows in set (0.00 sec)
```

在上述UPDATE语句中，将sh_goods表按价格升序排序，并限制更新库存量stock的记录只能是从第1条记录开始的两条记录。修改后可以从SELECT的查询结果中查看出价格最低的两种商品库存量stock已被修改为500，其余商品的stock值没有改变。

同样的，数据删除排序与限量的使用方式与数据更新的排序与限量相同，这里不再演示，感兴趣的读者可以尝试在不添加WHERE条件时，删除价格最高的两种商品。

5.3 分组与聚合函数

存储在数据库的海量数据，不仅可以根据项目需求实现数据的简单增、删、改、查操作，还可用于数据的统计分析，让每条数据变得更有价值。例如，电商网站根据用户的偏好(经常浏览/购买的商品种类)为其推荐最新最火爆的商品。在MySQL中提供分组操作的目的就是为了统计，其中为了方便统计还提供了大量的聚合函数。本节将针对MySQL中分组和聚合函数的使用进行详细讲解。

5.3.1 分组

在MySQL中，可以使用GROUP BY根据一个或多个字段进行分组，字段值相同的为一组。另外，对于分组的数据可以使用HAVING进行条件筛选。接下来为了便于读者理解，通过几种常用的方式对分组进行详细讲解。

1. 分组统计

在查询数据时，在WHERE条件后添加GROUP BY即可根据指定的字段进行分组，其基本语法格式如下。

```
SELECT [select 选项] 字段列表 FROM 数据表名
[WHERE 条件表达式] GROUP BY 字段名;
```

上述语法在MySQL 5.7中分组后，SELECT获取的字段列表只能是GROUP BY分组的字段，或使用了聚合函数的非分组字段，若在获取非分组字段时没有使用聚合函数，MySQL会报错误提示。

为了读者更好地理解，下面通过聚合函数 MAX()获取每个分类下商品的最高价格。具体 SQL 语句及执行结果如下。

```
mysql> SELECT category_id, MAX(price) FROM sh_goods GROUP BY category_id;
+-------------+------------+
| category_id | MAX(price) |
+-------------+------------+
|           3 |     15.00  |
|           6 |   1999.00  |
|           8 |     69.00  |
|           9 |    109.00  |
|          10 |   2000.00  |
|          12 |   5999.00  |
|          15 |    299.00  |
|          16 |     48.00  |
+-------------+------------+
8 rows in set (0.00 sec)
```

上述语句中，根据 category_id 进行分组，然后获取每个 category_id 分组下商品的最高价格。其中，MAX()是 MySQL 提供的一个聚合函数，用于获取 price 字段的最大值。

另外，在 MySQL 5.6 等老版本中，分组后获取的字段列表，若非分组字段没有使用聚合函数，默认情况下只保留每组中的第一条记录，但是此操作在 MySQL 5.7 及以上版本中已被禁止。那么，为了避免项目开发 MySQL 版本升级带来的问题，推荐读者在编写分组 SQL 语句时按照 MySQL 5.7 版本更严格的方式进行设计。

2. 分组排序

在 MySQL 中，默认情况下为分组操作的字段提供了升序排序的功能，因此在分组时可以为指定的字段进行升序或降序排序，其基本语法格式如下。

```
SELECT [select 选项] 字段列表 FROM 数据表名
[WHERE 条件表达式]
GROUP BY 字段名 [ASC | DESC];
```

需要注意的是，GROUP BY 分组排序的实现不需要使用 ORDER BY，直接在分组字段后添加 ASC(升序，默认值可省略)或 DESC(降序)即可。

下面根据 sh_goods 表中的分类 id 进行分组降序操作，查询并显示分组后每组的商品 id 以及商品的名称。具体 SQL 语句及执行结果如下。

```
mysql> SELECT category_id, GROUP_CONCAT(id), GROUP_CONCAT(name)
    -> FROM sh_goods GROUP BY category_id DESC;
+-------------+------------------+--------------------+
| category_id | GROUP_CONCAT(id) | GROUP_CONCAT(name) |
+-------------+------------------+--------------------+
|          16 | 10               | 薄毛衣             |
|          15 | 9                | 收腰风衣           |
```

```
|           12    | 4                   | 超薄笔记本         |
|           10    | 8                   | 办公电脑           |
|            9    | 7                   | 头戴耳机           |
|            8    | 6                   | 桌面音箱           |
|            6    | 5                   | 智能手机           |
|            3    | 1,2,3               | 2B铅笔,钢笔,碳素笔 |
+-------------+-----------------+------------------+-
8 rows in set (0.00 sec)
```

在上述语句中,聚合函数 GROUP_CONCAT()表示将指定字段值连接成一个字符串。例如,category_id 为 3 的分类下有 3 件商品,对应的 id 分别为 1、2 和 3。

3. 多分组统计

在对数据进行分组统计时,MySQL 中还支持数据按照某个字段进行分组后,对已经分组的数据进行再次分组的操作,以实现多分组统计。其基本语法格式如下。

```
SELECT [select 选项] 字段列表 FROM 数据表名
[WHERE 条件表达式]
GROUP BY 字段名 1 [ASC | DESC], [, 字段名 2 [ASC | DESC]]…;
```

在上述语法中,查询出的数据首先按照字段 1 进行分组排序,再将字段 1 相同的结果按照字段 2 进行分组排序,依此类推。

例如,对 sh_goods 表,以评分 score 降序分组后,再以评论数 comment_count 升序排序,获取的数据包括商品数量、指定分组下的商品名以及对应的评论数。具体 SQL 语句和执行结果如下。

```
mysql> SELECT score, COUNT(*), GROUP_CONCAT(name), comment_count
    -> FROM sh_goods GROUP BY score DESC, comment_count;
+-------+----------+-------------------+---------------+
| score | COUNT(*) | GROUP_CONCAT(name) | comment_count |
+-------+----------+-------------------+---------------+
|  5.00 |        2 | 碳素笔,智能手机    |         98000 |
|  4.90 |        2 | 2B铅笔,收腰风衣    |         40000 |
|  4.80 |        1 | 办公电脑           |          6000 |
|  4.80 |        1 | 薄毛衣             |         98000 |
|  4.50 |        1 | 桌面音箱           |          1000 |
|  3.90 |        2 | 钢笔,头戴耳机      |           500 |
|  2.50 |        1 | 超薄笔记本         |           200 |
+-------+----------+-------------------+---------------+
7 rows in set (0.00 sec)
```

从以上的执行结果可知,以 score 降序分组后,再以 comment_count 对查询的内容进行升序分组。例如,score 为 4.8 分组下第 1 件商品的 comment_count 为 6000,第 2 件商品的 comment_count 为 98000。

4. 回溯统计

回溯统计可以简单地理解为在根据指定字段分组后,系统又自动对分组的字段向上进

行了一次新的统计并产生一个新的统计数据，且该数据对应的分组字段值为 NULL。其基本语法格式如下。

```
SELECT [select 选项] 字段列表 FROM 数据表名
[WHERE 条件表达式]
GROUP BY 字段名 1 [ASC | DESC], [, 字段名 2 [ASC | DESC]]… WITH ROLLUP;
```

从上述语法可知，回溯统计的实现很简单，只需要在"GROUP BY 字段"后添加 WITH ROLLUP 即可。

但读者可能对回溯统计的作用还不能很好理解。下面以查看 sh_goods 表中每个分类 category_id 下的商品数量为例演示回溯统计的使用，具体 SQL 语句及执行结果如下。

```
mysql> SELECT category_id, COUNT(*)
    -> FROM sh_goods GROUP BY category_id WITH ROLLUP;
+-------------+----------+
| category_id | COUNT(*) |
+-------------+----------+
|           3 |        3 |
|           6 |        1 |
|           8 |        1 |
|           9 |        1 |
|          10 |        1 |
|          12 |        1 |
|          15 |        1 |
|          16 |        1 |
|        NULL |       10 |
+-------------+----------+
9 rows in set (0.00 sec)
```

从上述执行结果可知，在获取每种商品分类 category_id 下的商品数量后，系统又自动对获取的数量进行了一次累加统计，并且此累加的新数据（如 10）对应的分组字段（如 category_id）的值为 NULL。此行的记录就是对 category_id 分组的一次回溯统计。

在了解了单字段的分组回溯统计后，接下来演示如何在 MySQL 中对多分组进行回溯统计。具体 SQL 语句及执行结果如下。

```
mysql> SELECT score, comment_count, COUNT(*)
     -> FROM sh_goods
     -> GROUP BY score, comment_count WITH ROLLUP;
+-------+---------------+----------+
| score | comment_count | COUNT(*) |
+-------+---------------+----------+
|  2.50 |           200 |        1 |
|  2.50 |          NULL |        1 |
|  3.90 |           500 |        2 |
|  3.90 |          NULL |        2 |
```

```
| 4.50    |         1000 |        1 |
| 4.50    |         NULL |        1 |
| 4.80    |         6000 |        1 |
| 4.80    |        98000 |        1 |
| 4.80    |         NULL |        2 |
| 4.90    |        40000 |        2 |
| 4.90    |         NULL |        2 |
| 5.00    |        98000 |        2 |
| 5.00    |         NULL |        2 |
| NULL    |         NULL |       10 |
+-------- +------------+----------+
14 rows in set (0.00 sec)
```

在上述 SQL 语句中，分组操作根据 GROUP BY 后的字段从前往后依次执行，即先按 score 分组，然后再按 comment_count 分组；数据分组后系统再进行回溯统计，它与分组的操作正好相反，从 GROUP BY 后最后一个指定的分组字段开始回溯统计，并将结果上报，然后根据上报结果依次向前一个分组的字段进行回溯统计，即先回溯统计 comment_count 分组的结果，再根据 comment_count 的回溯结果对 score 分组进行回溯统计。

因此，从执行结果可看出，score 值相同的情况下，按 comment_count 分组后，实现了 6 次回溯统计，对应的分组字段 comment_count 含有 6 个 NULL 值，统计结果分别进行了累加；接着按上次的结果对 score 分组字段进行了 1 次回溯统计，对应的分组字段 score 中含有一个 NULL 值，然后又对上一次的统计结果再次进行累加。

值得一提的是，虽然回溯统计对数据的分析很有帮助，但是 MySQL 的同一个查询语句中回溯统计(WITH ROLLUP)与排序(ORDER BY)仅能出现一个。

5. 统计筛选

当对查询的数据执行分组操作时，可以利用 HAVING 根据条件进行数据筛选，它与前面学习过的 WHERE 功能相同，但是在实际运用时两者有一定的区别。

- WHERE 操作是从数据表中获取数据，将数据从磁盘存储到内存中，而 HAVING 是对已存放到内存中的数据进行操作。
- HAVING 位于 GROUP BY 子句后，而 WHERE 位于 GROUP BY 子句之前。
- HAVING 关键字后可以使用聚合函数，而 WHERE 则不可以。通常情况下，HAVING 关键字与 GROUP BY 一起使用，对分组后的结果进行过滤。

HAVING 统计筛选的基本语法格式如下。

```
SELECT [select 选项] 字段列表 FROM 数据表名
[WHERE 条件表达式]
GROUP BY 字段名 [ASC | DESC], … [WITH ROLLUP]
HAVING 条件表达式;
```

在上述的语法中，WHERE 条件之后的所有语句都是对内存中的数据进行操作。而 HAVING 则根据条件表达式对分组后的内容进行过滤。

例如，查询 sh_goods 表，获取评分 score 和评分数 comment_count 不同的情况下，含有两件商品的对应商品 id。具体 SQL 语句及执行结果如下。

```
mysql> SELECT score, comment_count, GROUP_CONCAT(id)
    -> FROM sh_goods
    -> GROUP BY score, comment_count
    -> HAVING COUNT(*) = 2;
+--------+---------------+------------------+
| score  | comment_count | GROUP_CONCAT(id) |
+--------+---------------+------------------+
|  3.90  |           500 | 2,7              |
|  4.90  |         40000 | 1,9              |
|  5.00  |         98000 | 3,5              |
+--------+---------------+------------------+
3 rows in set (0.00 sec)
```

在上述 SQL 语句中，首先根据评分（score）进行分组，然后再根据评分数（comment_count）进行分组，分组后利用 HAVING 筛选商品数量等于 2 的数据信息。

多学一招：在查询中使用别名

在 MySQL 中执行查询操作时，可以根据具体情况为获取的字段设置别名。例如，通过设置别名来缩短字段的名称长度。基本语法格式如下所示。

```
SELECT 字段 1 [AS] 别名 1, 字段 2 [AS] 别名 2 [, …] FROM 表名
```

在上述语法中，AS 用于为其前面的字段、表达式、函数等设置别名。也可以省略 AS 使用空格代替。例如，为 category_id 设置别名 cid，可以使用 category_id AS cid 或 category_id cid。

为了读者更好地理解，下面以获取商品分类 id 为 3 或 6 的商品的最高价格为例，演示别名的使用。具体 SQL 语句及执行结果如下。

```
mysql> SELECT category_id cid, MAX(price) max_price FROM sh_goods
    -> GROUP BY cid HAVING cid = 3 OR cid = 6;
+-----+-----------+
| cid | max_price |
+-----+-----------+
|  3  |    15.00  |
|  6  |  1999.00  |
+-----+-----------+
1 row in set (0.00 sec)
```

在上述语句中，为查询的字段 category_id 和 MAX(price)分别设置了别名 cid 和 max_price 后，在 GROUP BY 分组、HAVING 分组筛选和查询结果中就可以使用设置的别名，方便开发与阅读。

此外，在 MySQL 中还可以使用 AS 为数据表设置别名，基本语法格式如下。

```
SELECT 表别名.字段 [, …] FROM 表名 [AS] 表别名
```

同样地，在为数据表设置别名时，AS 也可以省略，使用空格代替。例如，为以上示例中的数据表设置别名，修改效果如下。

```
SELECT g.category_id cid, MAX(g.price) max_price FROM sh_goods g
GROUP BY cid HAVING cid = 3 OR cid = 6;
```

需要注意的是，字段与表设置别名后，在排序和分组中可以使用原来的字段名等，也可以使用别名。表的别名主要在多表查询中使用，具体会在后面章节讲解。

5.3.2 聚合函数

通过前面的学习可知，在对数据进行分组统计时，经常需要结合 MySQL 提供的聚合函数才能够统计出具有价值的数据。例如，获取每种商品分类下的商品数量、最高价格的商品、最低价格的商品等。为此，MySQL 中的聚合函数就可在查询数据时提供一些特殊的功能，具体如表 5-2 所示。

表 5-2 常用的聚合函数

函数名	描述
COUNT()	返回参数字段的数量，不统计为 NULL 的记录
SUM()	返回参数字段之和
AVG()	返回参数字段的平均值
MAX()	返回参数字段的最大值
MIN()	返回参数字段的最小值
GROUP_CONCAT()	返回符合条件的参数字段值的连接字符串
JSON_ARRAYAGG()	将符合条件的参数字段值作为单个 JSON 数组返回，MySQL 5.7.22 新增
JSON_OBJECTAGG()	将符合条件的参数字段值作为单个 JSON 对象返回，MySQL 5.7.22 新增

在表 5-2 中，COUNT()、SUM()、AVG()、MAX()、MIN()和 GROUP_CONCAT()函数中可以在参数前添加 DISTINCT，表示对不重复的记录进行相关操作。其中，COUNT()的参数设置为“*”时，表示统计符合条件的所有记录(包含 NULL)。

下面为了读者更好地理解，演示聚合函数单独获取指定表中符合条件的记录信息。具体如下。

```
mysql> SELECT MAX(price), MIN(price) FROM sh_goods;
+------------+------------+
| MIN(price)       | MAX(price)       |
+------------+------------+
|    5999.00       |       0.50       |
+------------+------------+
1 row in set (0.00 sec)
```

从上述的操作可知，利用 MAX()和 MIN()聚合函数可以从 sh_goods 表所有的记录中获取商品价格 price 最高和最低的值。

除此之外，聚合函数还可以与分组操作一起使用，用于分析分组后的数据信息。例如，在 sh_goods 中，获取不同 category_id 下商品数大于 2 的最高与最低的商品价格。具体

SQL 语句及执行结果如下。

```
mysql> SELECT category_id, MAX(price), MIN(price)
    -> FROM sh_goods GROUP BY category_id HAVING COUNT(*) > 2;
+-------------+------------+------------+
| category_id | MAX(price) | MIN(price) |
+-------------+------------+------------+
|           3 |      15.00 |       0.50 |
+-------------+------------+------------+
1 row in set (0.00 sec)
```

在上述语句中，首先根据 category_id 进行分组，获取每组中商品价格 price 的最大值与最小值，然后选取每组中商品数量大于 2 的分组，即可得到以上的结果。

除此之外，在 MySQL 5.7.22 中还新增了两个函数 JSON_ARRAYAGG()和 JSON_OBJECTAGG()，用于将符合条件的结果转为 JSON 格式。示例如下。

```
mysql> SELECT category_id, JSON_ARRAYAGG(id), JSON_OBJECTAGG(id, name)
    -> FROM sh_goods GROUP BY category_id;
+-------------+-------------------+-------------------------------------------+
| category_id | JSON_ARRAYAGG(id) | JSON_OBJECTAGG(id, name)                  |
+-------------+-------------------+-------------------------------------------+
|           3 | [1, 2, 3]         | {"1": "2B铅笔", "2": "钢笔", "3": "碳素笔"} |
|           6 | [5]               | {"5": "智能手机"}                          |
|           8 | [6]               | {"6": "桌面音箱"}                          |
|           9 | [7]               | {"7": "头戴耳机"}                          |
|          10 | [8]               | {"8": "办公电脑"}                          |
|          12 | [4]               | {"4": "超薄笔记本"}                        |
|          15 | [9]               | {"9": "收腰风衣"}                          |
|          16 | [10]              | {"10": "薄毛衣"}                           |
+-------------+-------------------+-------------------------------------------+
8 rows in set (0.00 sec)
```

在上述语句中，JSON_ARRAYAGG()函数的参数可以是一个字段或表达式，返回值是一个 JSON 数组；JSON_OBJECTAGG()函数用于返回 JSON 对象，第 1 个参数表示“键”，第 2 个参数表示“键”对应的值。

5.4　运算符

在数据库操作中，数据的 SELECT、UPDATE 和 DELETE 等操作都可以使用条件表达式，用于获取、更新或删除给定条件的数据。例如，获取商品数据表中价格在 2000 与 5000 之间的所有商品的打折信息。此时就需要使用 MySQL 提供的运算符才能完成用户的需求。本节将针对 MySQL 中运算符的使用进行详细讲解。

5.4.1　算术运算符

算术运算符适用于数值类型的数据，通常应用在 SELECT 查询结果的字段中，在 WHERE 条件表达式中应用较少，具体如表 5-3 所示。

表 5-3　算术运算符

运算符	描　述	示　　例	运算符	描　述	示　　例
＋	加运算	SELECT 5＋2；	/	除运算	SELECT 5/2；
－	减运算	SELECT 5－2；	%	取模运算	SELECT 5%2；
*	乘运算	SELECT 5 * 2；			

在表 5-3 中，运算符两端的数据可以是真实的数据（如 5），或数据表中的字段（如 price），而参与运算的数据一般称之为操作数，操作数与运算符组合在一起统称为表达式（如 5＋2）。另外，在 MySQL 中可以直接利用 SELECT 查看数据的运算结果，如表 5-3 中的示例。

算术运算符的使用看似简单，但是在实际应用时还有几点需要注意。下面为了让读者更好地理解，通过案例的方式一一进行演示。

1. 无符号的加减乘法运算

在 MySQL 中，若运算符"＋""－"和"*"的操作数都是无符号整型，则运算结果也是无符号整型。例如，商品表 sh_goods 中的 id 字段就是无符号整型，下面对 sh_goods 表中前 5 条记录的 id 进行加 1、减 1 以及乘 2 操作。具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, id+1, id-1, id*2 FROM sh_goods LIMIT 5;
+----+------+------+------+
| id | id+1 | id-1 | id*2 |
+----+------+------+------+
|  1 |    2 |    0 |    2 |
|  2 |    3 |    1 |    4 |
|  3 |    4 |    2 |    6 |
|  4 |    5 |    3 |    8 |
|  5 |    6 |    4 |   10 |
+----+------+------+------+
5 rows in set (0.00 sec)
```

从上述执行结果可知，sh_goods 表中的前 5 条记录的 id 值分别为 1 到 5，然后在此基础上实现了 id＋1、id－1 和 id * 2 的运算，运算结果依然都为无符号整型。

2. 有符号的减法运算结果

MySQL 中，默认情况下运算符"－"的操作数若都为无符号整型，则结果一定是无符号的整型，若操作数的差值为负数，那么系统就会报错。例如，对 sh_goods 表中前 5 条记录的 id 执行减 3 操作，具体 SQL 语句及执行结果如下。

```
mysql> SELECT id-3 FROM sh_goods LIMIT 5;
ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in '(`shop`.`sh_goods`.`id` - 3)'
```

从上述执行结果可知，减法运算的结果已经超出了无符号 BIGINT 的最大范围。因此，无论运算符"－"的操作数是否含有符号，若要获得一个有符号的运算结果，可以使用

CAST(… AS SIGNED)将无符号整型 id 强制转换为有符号的整型数据。将上述语句修改成以下形式。

```
mysql> SELECT CAST(id AS SIGNED) - 3 FROM sh_goods LIMIT 5;
+-------------------+
| CAST(id AS sIGNED) - 3   |
+-------------------+
|                 -2  |
|                 -1  |
|                  0  |
|                  1  |
|                  2  |
+-------------------+
5 rows in set (0.00 sec)
```

上述执行结果，“—2”和“—1”中的符号“—”，表示负号，它是一个一元操作符，仅有一个操作数，如2。而运算符“—”表示减法运算时，是一个二元操作符，有两个操作数，如 id—3 中的 id 和 3。

3. 含有精度的运算

算术运算除了可以对整数运算外，还可以对浮点数进行运算。在对浮点数进行加减运算时，运算结果中的精度(小数点后的位数)等于参与运算的操作数的最大精度。例如 1.2 ＋1.400，1.400 的精度最大为 3，则运算结果的精度就为 3；在对浮点数进行乘法运算时，运算结果中的精度，以参与运算的操作数的精度和为准。例如 1.2 * 1.400，1.2 的精度为 1，1.400 的精度为 3，则运算结果中的精度就为 4(1＋3)。

下面查询 sh_goods 表，获取 5 星好评的商品添加 850 件库存后的值，以及 75 折促销后的价格，具体 SQL 语句及执行结果如下。

```
mysql> SELECT name, price, stock, price * 0.75, stock + 850.00
    -> FROM sh_goods WHERE score = 5;
+-----------+---------+-------+------------+-------------+
| name      | price   | stock | price * 0.75 | stock+ 850.00   |
+-----------+---------+-------+------------+-------------+
| 碳素笔     |   1.00  |  500  |   0.7500   |   1350.00   |
| 智能手机   | 1999.00 |    0  | 1499.2500  |    850.00   |
+-----------+---------+-------+------------+-------------+
2 rows in set (0.00 sec)
```

上述执行结果中，price 的精度为 2，stock 没有精度。因此，price * 0.75 的运算结果精度为 4(2＋2)，如碳素笔折后价格的小数点后有 4 位；stock＋850.00 的运算结果精度为 2(0 与 2 比较 2 大)，如碳素笔添加库存后小数点后有 2 位。

4. “/”运算

运算符“/”在 MySQL 中用于除法操作，且运算结果使用浮点数表示，浮点数的精度等于被除数(“/”运算符左侧的操作数)的精度加上系统变量 div_precision_increment 设置的

除法精度增长值，读者可通过以下 SQL 语句查找其默认值。

```
mysql> SHOW VARIABLES LIKE 'div_precision_increment';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| div_precision_increment | 4     |
+-------------------------+-------+
1 row in set, 1 warning (0.00 sec)
```

从上述的执行结果可知，div_precision_increment 的默认值为 4。

例如，在活动日，库存量大于 200 的商品，75 折优惠后卖出了 4/5，查询活动后剩余的库存量。具体 SQL 语句及执行结果如下。

```
mysql> SELECT name, stock, stock/5 FROM sh_goods WHERE stock > 200;
+---------+-------+----------+
| name    | stock | stock/5  |
+---------+-------+----------+
| 2B铅笔  |  500  | 100.0000 |
| 钢笔    |  300  |  60.0000 |
| 碳素笔  |  500  | 100.0000 |
| 桌面音箱 |  750  | 150.0000 |
+---------+-------+----------+
2 rows in set (0.00 sec)
```

在上述语句中，首先从 sh_goods 表中获取库存量大于 200 的商品，然后再查询对应的商品名称、原库存量以及计算活动后剩余的 1/5 库存量。如上述执行结果所示，运算的结果精度为 4(stock 的精度 0＋除法精度增长的默认值为 4)。

值得一提的是，除法运算中除数如果为 0，则系统显示的执行结果为 NULL。

5. NULL 参与算术运算

在算术运算中，NULL 是一个特殊的值，它参与的算术运算结果均为 NULL。例如，使用 NULL 参与加减乘除运算，具体 SQL 语句及执行结果如下。

```
mysql> SELECT NULL+1, 3-NULL, 7*NULL, 2/NULL, NULL/3;
+--------+--------+--------+--------+--------+
| NULL+1 | 3-NULL | 7*NULL | 2/NULL | NULL/3 |
+--------+--------+--------+--------+--------+
|  NULL  |  NULL  |  NULL  |  NULL  |  NULL  |
+--------+--------+--------+--------+--------+
1 row in set (0.00 sec)
```

6. DIV 与 MOD 运算符

在 MySQL 中，运算符 DIV 与"/"都能实现除法运算，区别在于前者的除法运算结果会去掉小数部分，只返回整数部分。具体 SQL 语句及执行结果如下。

```
mysql> SELECT 8/5, 8 DIV 5, 0.6/1.2, 0.6 DIV 1.2;
+--------+---------+---------+-------------+
| 8/5    | 8 DIV 5 | 0.6/1.2 | 0.6 DIV 1.2 |
+--------+---------+---------+-------------+
| 1.6000 |       1 | 0.50000 |           0 |
+--------+---------+---------+-------------+
1 row in set (0.00 sec)
```

从上述的运算结果对比可知，除法操作运算符“/”的结果为浮点数，而 DIV 的结果为整数。例如，“8/5”的结果为 1.6000，将“/”替换为 DIV，则执行结果只保留了整数部分的 1。

另外，MySQL 中的运算符 MOD 与“%”功能相同，都用于取模运算。具体 SQL 语句及执行结果如下。

```
mysql> SELECT 8 MOD 5, -8 MOD 5, 8 MOD -5, -8 MOD -5;
+---------+----------+----------+-----------+
| 8 MOD 5 | -8 MOD 5 | 8 MOD -5 | -8 MOD -5 |
+---------+----------+----------+-----------+
|       3 |       -3 |        3 |        -3 |
+---------+----------+----------+-----------+
1 row in set (0.00 sec)
```

从上述的执行结果可知，取模运算结果的正负与被模数(%左边的操作数)的符号相同，与模数(%右边的操作数)符号无关。

关于算术运算，除了上面讲解的算术运算符外，MySQL 中还提供了很多进行数学运算的函数，常用的如表 5-4 所示。

表 5-4 常用数学函数

运 算 符	描 述
CEIL(x)	返回大于等于 x 的最小整数
FLOOR(x)	返回小于等于 x 的最大整数
FORMAT(x,y)	返回小数点后保留 y 位的 x(进行四舍五入)
ROUND(x[,y])	计算离 x 最近的整数；若设置参数 y，与 FORMAT(x,y)功能相同
TRUNCATE(x,y)	返回小数点后保留 y 位的 x(舍弃多余小数位，不进行四舍五入)
ABS(x)	获取 x 的绝对值
MOD(x,y)	求模运算，与 x%y 的功能相同
PI()	计算圆周率
SQRT(x)	求 x 的平方根
POW(x,y)	幂运算函数，计算 x 的 y 次方，与 POWER(x,y)功能相同
RAND()	默认返回 0 到 1 之间的随机数，包括 0 和 1

在表 5-4 中，RAND()函数用于返回 0 到 1 之间的随机数，若要获取指定区间(min≤num<max)内的随机数时，使用表达式 FLOOR(min+RAND() * (max−min))获取。

例如，获取大于等于 1 且小于 10 的任意一个随机整数，具体 SQL 语句如下。

```
mysql> SELECT FLOOR(1+RAND() * (10-1));
+------------------------+
| FLOOR(1+RAND() * (10-1))  |
+------------------------+
|                   7       |
+------------------------+
1 row in set (0.00 sec)
```

需要注意的是，以上的执行结果是 1 到 10(不包括 10)之间的随机数，每次执行都不同。若要获取相同的随机数，可以为 RAND()函数添加整数参数，具体 SQL 语句如下。

```
mysql> SELECT RAND(4);
+---------------------+
| RAND(4)             |
+---------------------+
| 0.15595286540310166 |
+---------------------+
1 row in set (0.00 sec)
mysql> SELECT RAND();
+---------------------+
| RAND()              |
+---------------------+
| 0.9753922950247231  |
+---------------------+
1 row in set (0.00 sec)
mysql> SELECT RAND(4);
+---------------------+
| RAND(4)             |
+---------------------+
| 0.15595286540310166 |
+---------------------+
1 row in set (0.00 sec)
```

从以上的操作可知，当为 RAND()设置参数后，则每次相同参数获取的随机值固定。

值得一提的是，RAND()函数还可与 ORDER BY 结合使用，用于随机获取指定的数据。例如，随机获取某一商品分类下的所有商品 id。具体 SQL 语句及执行结果如下。

```
mysql> SELECT category_id, GROUP_CONCAT(id) FROM sh_goods
    -> GROUP BY category_id ORDER BY RAND() LIMIT 1;
+-------------+------------------+
| category_id | GROUP_CONCAT(id) |
+-------------+------------------+
|           3 | 1,2,3            |
+-------------+------------------+
1 row in set (0.00 sec)
```

上述 SQL 语句，根据商品分类进行分组，然后使用 ORDER BY RAND()实现分组后记录的随机排序，“LIMIT 1”用于限定获取的记录仅为一条。

5.4.2 比较运算符

比较运算符是 MySQL 常用运算符之一，通常应用在条件表达式中对结果进行限定。MySQL 中比较运算符的结果值有 3 种，分别为 1（TRUE，表示为真）、0（FALSE，表示为假）或 NULL。具体如表 5-5 所示。

表 5-5 比较运算符

运 算 符	描 述
=	用于相等比较
<=>	可以进行 NULL 值比较的相等运算符
>	表示大于比较
<	表示小于比较
>=	表示大于等于比较
<=	表示小于等于比较
<>、!=	表示不等于比较
BETWEEN…AND…	比较一个数据是否在指定的闭区间范围内，若在则返回 1，若不在则返回 0
NOT BETWEEN…AND…	比较一个数据是否不在指定的闭区间范围内，若不在则返回 1，若在则返回 0
IS	比较一个数据是否是 TRUE、FALSE 或 UNKNOWN，若是则返回 1，否则返回 0
IS NOT	比较一个数据是否不是 TRUE、FALSE 或 UNKNOWN，若不是则返回 1，否则返回 0
IS NULL	比较一个数据是否是 NULL，若是则返回 1，否则返回 0
IS NOT NULL	比较一个数据是否不是 NULL，若不是则返回 1，否则返回 0
LIKE '匹配模式'	获取匹配到的数据
NOT LIKE '匹配模式'	获取匹配不到的数据

比较运算符的使用看似简单，但是在实际应用时还有几点需要注意。下面为了让读者更好地理解，通过案例的方式一一进行演示。

1. 数据类型自动转换

表 5-5 中的所有运算符都可以对数字和字符串进行比较，若参与比较的操作数的数据类型不同，则 MySQL 会自动将其转换为同类型的数据后再进行比较。具体 SQL 语句及运算结果如下。

```
mysql> SELECT 5>='5', 3.0<>3;
+---------+---------+
| 5>='5'  | 3.0<>3  |
+---------+---------+
|      1  |      0  |
```

```
+---------+---------+
1 row in set (0.00 sec)
```

从以上执行结果可知，整数5与字符型5在比较时首先转换成相同类型，然后再比较5大于等于5，因此结果为1(表示真)。同理，3.0与3进行不相等的比较，当操作数不相等时，返回1，相等返回0(表示假)。

2. 比较结果为NULL

MySQL中比较运算符=、>、<、>=、<=、<>、!=在与NULL进行比较时，结果均为NULL。具体SQL语句及执行结果如下。

```
mysql> SELECT 0 = NULL, NULL < 1, NULL <> 2;
+----------+--------+---------+
| 0 = NULL | NULL<1 | NULL<>2 |
+----------+--------+---------+
|   NULL   |  NULL  |  NULL   |
+----------+--------+---------+
1 row in set (0.00 sec)
```

从上述的执行结果可知，当操作数为NULL时，运算符"=""<"和"<>"的执行结果均为NULL。读者可按以上方式测试其他运算符的比较结果为NULL的情况，这里不再演示。

3. "="与"<=>"的区别

在MySQL中运算符"="与"<=>"均可以用于比较数据是否相等，两者的区别在于后者可以对NULL值进行比较。具体SQL语句及执行结果如下。

```
mysql> SELECT NULL=NULL, NULL=1, NULL <=> NULL, NULL<=>1;
+-----------+--------+---------------+----------+
| NULL=NULL | NULL=1 | NULL <=> NULL | NULL<=>1 |
+-----------+--------+---------------+----------+
|   NULL    |  NULL  |             1 |        0 |
+-----------+--------+---------------+----------+
1 row in set (0.00 sec)
```

从上述执行结果可知，运算符"<=>"在比较两个NULL是否相等时返回值为1，比较NULL与1是否相等时返回0，而运算符"="的结果全部为NULL。

4. BETWEEN…AND…

在条件表达式中若需要对指定区间的数据进行判断时，可使用BETWEEN…AND…实现，基本语法格式如下。

```
BETWEEN 条件1 AND 条件2
```

在上述语法，用于表示条件1到条件2之间的范围(包含条件1和条件2)，并且在设置时，条件1必须小于等于条件2。例如，获取sh_goods表中价格在2000到6000的商品，商

品信息包括 id、name 和 price。具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, name, price FROM sh_goods
    -> WHERE price BETWEEN 2000 AND 6000;
+----+------------+---------+
| id | name       | price   |
+----+------------+---------+
|  4 | 超薄笔记本 | 5999.00 |
|  8 | 办公电脑   | 2000.00 |
+----+------------+---------+
2 rows in set (0.00 sec)
```

从上述执行结果可知，价格等于 2000 的商品也包含在“BETWEEN 2000 AND 6000”指定的区间范围内。

值得一提的是，NOT BETWEEN…AND…的使用方式与 BETWEEN…AND 相同，但是表示的含义正好相反。例如，使用 NOT BETWEEN…AND…修改上述示例。具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, name, price FROM sh_goods
    -> WHERE price NOT BETWEEN 2000 AND 6000;
+----+----------+---------+
| id | name     | price   |
+----+----------+---------+
|  1 | 2B 铅笔  |    0.50 |
|  2 | 钢笔     |   15.00 |
|  3 | 碳素笔   |    1.00 |
|  5 | 智能手机 | 1999.00 |
|  6 | 桌面音箱 |   69.00 |
|  7 | 头戴耳机 |  109.00 |
|  9 | 收腰风衣 |  299.00 |
| 10 | 薄毛衣   |   48.00 |
+----+----------+---------+
8 rows in set (0.00 sec)
```

从上述执行结果可知，利用“NOT BETWEEN 2000 AND 6000”可获取价格不在 2000 到 6000 范围内的所有商品。

5. IS NULL 与 IS NOT NULL

在条件表达式中若需要判断字段是否为 NULL，可以使用 MySQL 专门提供的运算符 IS NULL 或 IS NOT NULL。例如，获取 sh_goods 表中关键词不为空且价格最高的两件商品 id、name、price 和 keyword。具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, name, price, keyword FROM sh_goods
    -> WHERE keyword IS NOT NULL
    -> ORDER BY price DESC LIMIT 2;
```

```
+----+----------+---------+---------+
| id | name     | price   | keyword |
+----+----------+---------+---------+
|  4 | 超薄笔记本 | 5999.00 | 电子产品 |
|  8 | 办公电脑   | 2000.00 | 电子产品 |
+----+----------+---------+---------+
2 rows in set (0.00 sec)
```

从上述 SQL 语句可知，判断哪个字段不为空，只需在 IS NOT NULL 前添加对应的字段名即可。同理，IS NULL 与 IS NOT NULL 的使用方式相同，用于判断字段为空。

6. LIKE 与 NOT LIKE

LIKE 运算符在前面学习查看数据表时已讲解，它的作用就是模糊匹配，NOT LIKE 的使用方式与之相同，用于获取匹配不到的数据。

例如，在 sh_goods 表中，获取商品名称中含有"笔"的商品 id、name、price 和 content。具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, name, price, content FROM sh_goods
    -> WHERE name LIKE '%笔%';
+----+-----------+---------+-----------+
| id | name      | price   | content   |
+----+-----------+---------+-----------+
|  1 | 2B 铅笔    |    0.50 | 考试专用   |
|  2 | 钢笔       |   15.00 | 练字必不可少 |
|  3 | 碳素笔     |   1 .00 | 平时使用   |
|  4 | 超薄笔记本  | 5999.00 | 轻小便携   |
+----+-----------+---------+-----------+
4 rows in set (0.00 sec)
```

在上述 SQL 语句中，匹配模式符"%"可以匹配任意 0 到多个字符，因此执行结果为 4 条记录；若将其修改为"_"表示匹配任意 1 个字符，上述示例的执行结果就为空，没有符合要求的记录。因此，读者在使用时需根据实际的需求选择使用哪种匹配模式符。

多学一招：正则匹配查询

MySQL 中查询数据时，除了可以使用 LIKE 实现模糊查询，还可以利用 REGEXP 关键字指定正则匹配模式轻松完成更为复杂的查询。其中，正则表达式的语法与其他编程语言相同，读者可自行查看相关的资料进行学习，这里不再赘述。

例如，获取 sh_goods 表中描述字段内含有"人"或"必备"词语的商品 id、name 和 content 字段内容，具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, name, content FROM sh_goods
    -> WHERE content REGEXP '人|必备';
+----+--------+-------------+
| id | name   | content     |
```

```
+----+--------+--------------+
|  5  | 智能手机  | 人人必备        |
|  7  | 头戴耳机  | 独享个人世界     |
| 10  |薄毛衣    | 居家旅行必备     |
+----+--------+--------------+
3 rows in set (0.00 sec)
```

在上述 SQL 语句中，REGEXP 关键字用于标识正则匹配，“人|必备”为正则匹配模式。其中，符号“|”在正则中表示分隔符，用于分隔多种条件，在匹配时只要指定字段满足分隔符左右两边条件中的一个，就表示匹配成功。

关于比较运算，除了上面讲解的比较运算符外，MySQL 中还提供了很多进行比较运算的函数，常用的如表 5-6 所示。

表 5-6　比较函数

函　数	描　述
IN()	比较一个值是否在一组给定的集合内
NOT IN()	比较一个值是否不在一组给定的集合内
GREATEST()	返回最大的一个参数值，至少两个参数
LEAST()	返回最小的一个参数值，至少两个参数
ISNULL()	测试参数是否为空
COALESCE()	返回第一个非空参数
INTERVAL()	返回小于第一个参数的参数索引
STRCMP()	比较两个字符串

在表 5-6 中，函数 GREATEST()和 LEAST()的参数至少有两个，用于比较后返回一个最大或最小的参数值。IN()只要比较的字段或数据在给定的集合内，那么比较结果就为真，NOT IN()正好与 IN()的功能相反。

为了读者更好地理解，下面获取 sh_goods 表中 category_id 为 3 或 15 的商品 id、name、keyword 和 category_id。具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, name, keyword, category_id FROM sh_goods
    -> WHERE category_id IN(3, 15);
+----+----------+---------+-------------+
| id | name     | keyword | category_id |
+----+----------+---------+-------------+
|  1 | 2B 铅笔  | 文具    |      3      |
|  2 | 钢笔     | 文具    |      3      |
|  3 | 碳素笔   | 文具    |      3      |
|  9 | 收腰风衣 | 服装    |     15      |
+----+----------+---------+-------------+
4 rows in set (0.00 sec)
```

从上述执行结果可知，category_id 只要符合 IN()函数给定的集合(3, 15)中的任意一

个值，那么比较结果都为真。

5.4.3 逻辑运算符

逻辑运算符也是MySQL常用运算符之一，通常应用在条件表达式中的逻辑判断，与比较运算符结合使用。参与逻辑运算的操作数以及逻辑判断的结果只有3种，分别为1（TRUE，表示为真）、0（FALSE，表示为假）或NULL，具体如表5-7所示。

表5-7 逻辑运算符

运算符	描述
AND或&&	逻辑与。操作数全部为真，则结果为1；否则为0
OR或||	逻辑或。操作数中只要有一个为真，则结果为1；否则为0
NOT或!	逻辑非。操作数为0，则结果为1；操作数为1，则结果为0
XOR	逻辑异或。操作数一个为真，一个为假，则结果为1；若操作数全部为真或全部为假，则结果为0

在表5-7中，仅有逻辑非（NOT或!）是一元运算符，其余均为二元运算符。另外，NOT和“!”虽然功能相同，但是在一个表达式中同时出现时，先运算“!”，再运算“NOT”。

为让读者更好地理解，下面通过案例的方式演示逻辑运算符的使用及其注意事项。

1. 逻辑与

查询sh_goods表中关键词为“电子产品”的5星商品，信息包括商品的id、name和price。具体SQL语句及执行结果如下。

```
mysql> SELECT id, name, price FROM sh_goods
    -> WHERE keyword = '电子产品' && score = 5;
+----+----------+---------+
| id | name     | price   |
+----+----------+---------+
|  5 | 智能手机 | 1999.00 |
+----+----------+---------+
1 row in set (0.00 sec)
```

在上述SQL语句中，只有同时满足keyword等于电子产品，score等于5的记录才会被查询出来。

值得一提的是，在开发时，若使用“&&”连接多个相等比较的条件时，可以使用(a,b)=(x,y)的方式简化条件表达式(a=x && b=y)的书写。使用此方式修改上述示例，具体SQL语句如下所示。

```
SELECT id, name, price FROM sh_goods
WHERE (keyword, score) = ('电子产品', 5);
```

另外，在进行逻辑与操作时，若操作数中含有NULL，而另一个操作数若为1(真)，则结果为NULL；若另一个操作数为0(假)，则结果为0。具体SQL语句如下。

```
mysql> SELECT 1&&NULL, NULL&&1, 0&&NULL, NULL&&0;
+----------+----------+----------+----------+
| 1&&NULL  | NULL&&1  | 0&&NULL  | NULL&&0  |
+----------+----------+----------+----------+
|   NULL   |   NULL   |     0    |     0    |
+----------+----------+----------+----------+
1 row in set (0.00 sec)
```

2. 逻辑或

查询 sh_goods 表中评分为 4.5 或价格小于 10 的商品,信息包括商品的 id、name、price 和 score。具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, name, price, score FROM sh_goods
    -> WHERE score = 4.5 || price < 10;
+----+----------+-------+-------+
| id | name     | price | score |
+----+----------+-------+-------+
|  1 | 2B 铅笔  |  0.50 |  4.90 |
|  3 | 碳素笔   |  1.00 |  5.00 |
|  6 | 桌面音箱 | 69.00 |  4.50 |
+----+----------+-------+-------+
3 rows in set (0.00 sec)
```

在上述 SQL 语句中,只要商品记录满足 score 等于 4.5,或 price 小于 10 中任意一个条件,数据就会被查询出。

另外,在进行逻辑或操作时,若操作数中含有 NULL,而另一个操作数若为 1(真),则结果为 1;若另一个操作数为 0(假),则结果为 NULL。具体 SQL 语句如下。

```
mysql> SELECT 1||NULL, NULL||1, 0||NULL, NULL||0;
+----------+----------+----------+----------+
| 1||NULL  | NULL||1  | 0||NULL  | NULL||0  |
+----------+----------+----------+----------+
|     1    |     1    |   NULL   |   NULL   |
+----------+----------+----------+----------+
1 row in set (0.00 sec)
```

3. 逻辑非

逻辑非的操作数仅有一个,当操作数为 0(假)时,则运算结果为 1;当操作数为 1(真)时,则运算结果为 0;当操作数为 NULL 时,运算结果为 NULL。具体 SQL 语句及执行结果如下。

```
mysql> SELECT NOT 10, NOT 0, NOT NULL, NOT 0 + !0, !0 + !0;
+--------+-------+----------+------------+---------+
| NOT 10 | NOT 0 | NOT NULL | NOT 0 + !0 | !0+ !0  |
```

```
+---------+-------+----------+------------+---------+
|    0    |   1   |   NULL   |      0     |    2    |
+---------+-------+----------+------------+---------+
1 row in set (0.00 sec)
```

上述 SQL 语句中,逻辑非运算符“!”的优先级别最高,其次为算术运算符“+”,优先级别最低的为 NOT。因此,表达式“NOT 0 + !0”首先计算“!0”结果为 1,然后计算“0+1”,结果为 1,最后计算“!1”,得到结果为 0。而表达式“!0 + !0”首先计算“!0”结果为 1,然后计算“1+1”,得到结果为 2。所以,读者在进行逻辑非运算时,要根据实际需求正确选择使用“!”或 NOT。

4. 逻辑异或

逻辑异或操作,表示两个操作数同时都为 1 或 0,则结果为 0;若两个操作数一个为 1,一个为 0,则结果为 1;当操作数为 NULL 时,则结果为 NULL。具体 SQL 语句及执行结果如下。

```
mysql> SELECT 1 XOR 2, 0 XOR 0, 0 XOR 2, NULL XOR 2;
+---------+---------+---------+------------+
| 1 XOR 2 | 0 XOR 0 | 0 XOR 2 | NULL XOR 2 |
+---------+---------+---------+------------+
|    0    |    0    |    1    |    NULL    |
+---------+---------+---------+------------+
1 row in set (0.00 sec)
```

5.4.4 赋值运算符

MySQL 中“=”是一个比较特殊的运算符,既可以用于比较数据是否相等,又可以表示赋值。因此,MySQL 为了避免系统分不清楚运算符“=”表示赋值还是比较的含义,特意增加一个符号“:=”用于表示赋值运算。

例如,查看 sh_goods 表中评分为 4.5 的商品 id、name 和 stock,具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, name, stock FROM sh_goods WHERE score = 4.5;
+----+----------+-------+
| id | name     | stock |
+----+----------+-------+
|  6 | 桌面音箱 |  750  |
+----+----------+-------+
1 row in set (0.00 sec)
```

在上述 SQL 语句中,WHERE 后的运算符“=”用于比较 sh_goods 表中 score 字段的值是否与 4.5 相等,若相等,则返回查询的记录。

接下来,在促销活动时,将评分等于 4.5 的商品库存修改为 1000。具体 SQL 语句。

```
UPDATE sh_goods SET stock = 1000 WHERE score = 4.5;
```

在上述 UPDATE 语句中，“stock＝1000”中的运算符“＝”表示赋值的含义，将符合 WHERE 比较条件的记录对应的 stock 字段值设置为 1000。因此，“stock＝1000”中的“＝”可使用 MySQL 提供的专门用于赋值的运算符“:=”代替，修改后 SQL 语句如下。

```
UPDATE sh_goods SET stock :=1000 WHERE score =4.5;
```

值得一提的是，在 MySQL 中，INSERT…SET 和 UPDATE…SET 中出现的运算符“=”都会被认为是赋值运算符。因此，建议除此之外的其他情况，若需要赋值运算符，推荐使用“:=”，如为变量赋值，后面的章节会详细地讲解，此处读者了解即可。

5.4.5　位运算符

位运算符是针对二进制数的每一位进行运算的符号，运算的结果类型为 BIGINT，最大范围可以是 64 位。具体如表 5-8 所示。

表 5-8　位运算符

运　算　符	描　　述	示　　例
&	按位与	SELECT b'1001' & b'1011';　　结果为 9
|	按位或	SELECT b'1001'| b'1011';　　结果为 11
^	按位异或	SELECT b'1001'^ b'1011';　　结果为 2
<<	按位左移	SELECT b'1001'<<2;　　结果为 36
>>	按位右移	SELECT b'1001'>>2;　　结果为 2
～	按位取反	SELECT ～b'1001'& b'1011';　结果为 2

需要注意的是，MySQL 5.7 中参与位运算的数据只能是 BIGINT 类型(64 位的整数)，而在 MySQL 8.0 中则允许二进制字符串类型的参数，如 BINARY、VARBINARY 和 BLOB。因此，MySQL 5.7 中二进制类型字段的位运算可能在 MySQL 8.0 中产生不同结果，系统也会报相关的警告信息。下面以一个案例演示 VARBINARY 类型的数据位运算。

```
#① 创建数据表,含有 VARBINARY 类型的字段
mysql> CREATE TABLE mydb.mybin (b1 VARBINARY(20), b2 VARBINARY(20));
Query OK, 0 rows affected (0.01 sec)
#② 插入数据
mysql> INSERT INTO mydb.mybin VALUES (2, 6), (3, 1), (4, 9);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
#③ 查看数据表中两个字段的按位与、按位或结果
mysql> SELECT b1&b2, b1|b2 FROM mydb.mybin;
+-------+-------+
| b1&b2   | b1|b2   |
```

```
+-------+-------+
|   2   |   6   |
|   1   |   3   |
|   0   |  13   |
+-------+-------+
3 rows in set,2 warnings (0.00 sec)
```

从上述操作可知，在对 VARBINARY 类型的字段进行位运算时，产生了两个警告行，使用 SHOW WARNINGS\G 查看，具体信息如下。

```
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1287
Message: Bitwise operations on BINARY will change behavior in a future version, check the 'Bit functions
' section in the manual.
*************************** 2. row ***************************
  Level: Warning
   Code: 1287
Message: Bitwise operations on BINARY will change behavior in a future version, check the 'Bit functions
' section in the manual.
2 rows in set (0.00 sec)
```

为了解决这个问题，可以在进行位运算时，使用 CAST(… AS UNSIGNED)将二进制类型的数据显式地转换为 BIGINT 类型。

```
mysql> SELECT
    -> CAST(b1 AS UNSIGNED) & CAST(b2 AS UNSIGNED) AS one,
    -> CAST(b1 AS UNSIGNED) | CAST(b2 AS UNSIGNED) AS two
    -> FROM mydb.mybin;
+-----+-----+
| one | two |
+-----+-----+
|  2  |  6  |
|  1  |  3  |
|  0  | 13  |
+-----+-----+
3 rows in set (0.00 sec)
```

在上述 SQL 语句中，将参与位运算的字段转换为整型后，运算的结果不再有警告信息。为方便查看与显示，使用 AS 为位运算表达式设置了别名，分别是 one 和 two。

关于位运算，除了上面讲解的位运算符外，MySQL 中还提供了进行位运算的函数，常用的如表 5-9 所示。

表 5-9 位运算相关函数

函 数	描 述	示 例
BIT_COUNT(N)	返回在参数 N 中设置的比特位(二进制位为 1)的数量	SELECT BIT_COUNT(b'1011'); 结果为 3
BIT_AND()	按位返回与的结果	SELECT BIT_AND(b1) FROM btable; 结果为 0

续表

函　　数	描　　述	示　　例
BIT_OR()	按位返回或的结果	SELECT BIT_OR(b1) FROM btable;结果为 7
BIT_XOR()	按位返回异或的结果	SELECT BIT_XOR(b1) FROM btable;结果为 5

5.4.6　运算符优先级

运算符优先级可以理解为运算符在一个表达式中参与运算的先后顺序,优先级别越高,则越早参与运算;优先级别越低,则越晚参与运算。下面看一下 MySQL 中所有运算符从高到低的优先级,如表 5-10 所示。

表 5-10　运算符优先级

<table>
<tr><th>运　算　符</th></tr>
<tr><td>INTERVAL</td></tr>
<tr><td>BINARY、COLLATE</td></tr>
<tr><td>!</td></tr>
<tr><td>-(一元,负号)、~(一元,按位取反)</td></tr>
<tr><td>^</td></tr>
<tr><td>*、/、DIV、%、MOD</td></tr>
<tr><td>-(相减运算符号)、+</td></tr>
<tr><td><<、>></td></tr>
<tr><td>&</td></tr>
<tr><td>|</td></tr>
<tr><td>= (比较运算符)、<=>、>=、>、<=、<、<>、!=、IS、LIKE、REGEXP、IN</td></tr>
<tr><td>BETWEEN、CASE、WHEN、THEN、ELSE</td></tr>
<tr><td>NOT</td></tr>
<tr><td>AND、&&</td></tr>
<tr><td>XOR</td></tr>
<tr><td>OR、||</td></tr>
<tr><td>= (赋值运算符)、:=</td></tr>
</table>

在表 5-10 中,同行的运算符具有相同的优先级,除赋值运算符从右到左运算外,其余相同级别的运算符,在同一个表达式中出现时,运算的顺序为从左到右依次进行。

除此之外,若要提升运算符的优先级别,可以使用圆括号"()",当表达式中同时出现多个圆括号时,最内层的圆括号中的表达式优先级最高。具体 SQL 语句及执行结果如下。

```
mysql> SELECT 2+3 * 5, (2+3) * 5;
+--------+----------+
| 2+3 * 5  | (2+3) * 5  |
+--------+----------+
|    17    |      25    |
+--------+----------+
1 row in set (0.00 sec)
```

在上述SQL语句中，表达式“2＋3 * 5”首先按运算符的优先级，计算乘法，然后再计算加法，因此结果为17；而表达式“(2＋3) * 5”则先计算圆括号内的加法，然后再计算乘法，因此结果为25。

在实际开发中，为复杂的表达式适当地添加圆括号，让编写的SQL语句更为清楚，避免因不清楚运算符优先级顺序而导致运算发生问题。

5.5 动手实践：商品评论表的操作

数据库的学习在于多看、多学、多想、多动手，只有将理论与实际相结合，才能够体现出数据开发与管理的重要性，展现知识学习的价值与力量。接下来请结合本章所学的知识完成商品评论表的操作。

【实践目标】

此实践的目标就是能够根据文字提示，完成商品表(sh_goods)与商品评论表(sh_goods_comment)各种需求的查询操作。

【实践需求】

(1) 查询商品id等于8且有效的评论内容。

(2) 查询每个用户评论的商品数量。

(3) 查询最新发布的5条有效商品评论信息。

(4) 查询评论过两种以上不同商品的用户id及对应的商品id。

(5) 结合sh_goods和sh_goods_comment表，查询没有任何评论信息的商品id和name。

(6) 结合sh_goods和sh_goods_comment表，查询商品评分为5星的商品评论信息。

【动手实践】

(1) 查询商品id等于8且有效的评论内容。

利用比较运算符“=”和逻辑运算符“&&”完成指定需求记录的查询，具体SQL语句及执行结果如下。

```
mysql> SELECT id, content FROM sh_goods_comment
    -> WHERE goods_id = 8 && is_show = 1;
```

```
+-----+----------+
| id  | content  |
+-----+----------+
|  6  | 卡机     |
+-----+----------+
1 row in set (0.00 sec)
```

（2）查询每个用户评论的商品数量。

由于一个用户可以评论多件商品，因此，若要查询每个用户评论的商品数量，首先需要根据用户 id 进行分组，然后再利用聚合函数 COUNT()获取每个分组下商品的数量。具体 SQL 语句及执行结果如下。

```
mysql> SELECT user_id, COUNT(goods_id) FROM sh_goods_comment
    -> GROUP BY user_id;
+----------+-----------------+
| user_id  | COUNT(goods_id) |
+----------+-----------------+
|        1 |               1 |
|        2 |               1 |
|        3 |               1 |
|        4 |               3 |
|        5 |               1 |
|        6 |               1 |
|        7 |               1 |
|        8 |               1 |
+----------+-----------------+
8 rows in set (0.00 sec)
```

从上述执行结果可以看出，只有 user_id 等于 4 的用户评论过 3 件商品，其他用户都只评论过 1 件商品。

（3）查询最新发布的 5 条有效商品评论信息。

根据文字描述可知，查询的评论信息，首先是有效的，然后是最新发布的 5 条记录。具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, content, user_id, goods_id
    -> FROM sh_goods_comment
    -> WHERE is_show = 1
    -> ORDER BY create_time DESC
    -> LIMIT 5;
+-----+-----------------+----------+----------+
| id  | content         | user_id  | goods_id |
+-----+-----------------+----------+----------+
| 10  | 特别彰显气质    |        8 |        9 |
|  9  |还行             |        4 |        9 |
|  7  | 黑夜拍照也清晰  |        6 |        5 |
|  6  | 卡机            |        5 |        8 |
|  4  | 携带方便        |        4 |        4 |
+-----+-----------------+----------+----------+
5 rows in set (0.00 sec)
```

(4) 查询评论过两种以上不同商品的用户 id 及对应的商品 id。

根据文字提示可知,首先需要根据用户 id 进行分组,然后再利用 HAVING 执行查询的需求,限定评论的商品数量必须大于等于 2,最后获取用户 id 和该用户评价过的商品 id。具体 SQL 语句及执行结果如下。

```
mysql> SELECT user_id, GROUP_CONCAT(goods_id)
    -> FROM sh_goods_comment
    -> GROUP BY user_id
    -> HAVING COUNT(DISTINCT goods_id) >= 2;
+---------+------------------------+
| user_id | GROUP_CONCAT(goods_id) |
+---------+------------------------+
|       4 | 4,7,9                  |
+---------+------------------------+
1 row in set (0.00 sec)
```

(5) 结合 sh_goods 和 sh_goods_comment 表,查询没有任何评论信息的商品 id 和 name。

根据文字提示可知,首先从 sh_goods 表中查询 id 和 name,然后使用 WHERE 指定查询的需求,利用 NOT IN()判断 id 是否在 sh_goods_comment 表查询出的商品 id 内。具体 SQL 语句及执行结果如下。

```
mysql> SELECT id,name FROM sh_goods WHERE id
    -> NOT IN(SELECT DISTINCT goods_id FROM sh_goods_comment);
+----+----------+
| id | name     |
+----+----------+
|  1 | 2B 铅笔  |
|  2 | 钢笔     |
|  3 | 碳素笔   |
|  6 | 桌面音箱 |
+----+----------+
4 rows in set (0.00 sec)
```

(6) 结合 sh_goods 和 sh_goods_comment 表,查询商品评分为 5 星的商品评论信息。

从 sh_goods_comment 表中获取评论信息,然后使用 WHERE 指定查询的条件,利用 IN()判断 goods_id 是否在 sh_goods 表中查询出的 5 星商品 id 内。具体 SQL 语句及执行结果如下。

```
mysql> SELECT id,content FROM sh_goods_comment
    -> WHERE goods_id IN(SELECT id FROM sh_goods WHERE score = 5);
+----+----------------+
| id | content        |
+----+----------------+
|  7 | 黑夜拍照也清晰 |
+----+----------------+
1 row in set (0.00 sec)
```

5.6　本章小结

本章主要讲解了如何根据一张已有的数据表创建新表，以及数据复制的插入，数据查询、更新、删除的排序与限量，数据查询的分组以及如何通过运算符完成条件表达式的编写。其中，单表的查询操作是本章需要重点掌握的内容。

5.7　课后练习

一、填空题

1. 多数据插入时，VALUE 后的多个值列表之间使用________分隔。
2. “LIMIT 2,2”表示从第________条记录开始，最多获取 2 条记录。
3. 表达式________用于获取大于等于 3 且小于等于 11 之间的随机数。
4. MySQL 的除法运算中，除数为 0 的执行结果为________。
5. 在 INSERT 语句中添加________可在主键冲突时，利用更新的方式完成数据的插入。

二、判断题

1. 查询数据时，默认根据 ORDER BY 指定的字段进行降序排序。(　　)
2. UPDATE 更新数据时可以通过 LIMIT 限制更新的记录数。(　　)
3. “LIMIT 3”中的 3 表示偏移量，用于设置从哪条记录开始。(　　)
4. 使用 SELECT 查看表达式“NOT 2 ＋ !3”的执行结果为 0。(　　)
5. 对于分组数据的排序，只需在分组字段后添加 ASC 或 DESC 即可。(　　)

三、选择题

1. 下面关于插入数据的语法错误的是(　　)。
   A. INSERT INTO 表 VALUE(值列表)；
   B. INSERT 表 SET 字段值 1＝值 1[，字段 2＝值 2]…；
   C. INSERT INTO 表 1(字段列表) SELECT (字段列表) FROM 表 2；
   D. INSERT INTO 表 1(字段列表) VALUES SELECT (字段列表) FROM 表 2；
2. 下列选项中与“WHERE (id，price)＝(3，1999)”功能相同的是(　　)。
   A. WHERE id＝3 || price＝1999　　B. WHERE id＝3 && price＝1999
   C. WHERE (id，price)<> (3，1999)　　D. 以上选项都不正确
3. 以下可以用于比较运算的函数是(　　)。
   A. RAND()　　B. POW()　　C. CEIL()　　D. IN()
4. 以下运算符中，优先级别最高的是(　　)。
   A. －(负号)　　B. －(减运算符)
   C. ＝(赋值运算符)　　D. ＝(比较运算符)

5. 下面关于分组的说法错误的是(　　)。
   A. SELECT 语句中 ORDER BY 不能与回溯统计同时使用
   B. 利用 ANY_VALUE()可使分组统计后默认只保留每组中的第一条记录
   C. 分组后的数据筛选可以使用 WHERE 或 HAVING 实现
   D. 分组操作默认按分组字段(中文除外)升序排序

四、简答题

1. 请简述 DELETE 与 TRUNCATE 的区别。
2. 请简述 WHERE 与 HAVING 之间的区别。

五、实训题

1. 依据 sh_goods 的结构与数据,在 mydb 数据库中创建一张 tm_goods 表,并将价格在 20 到 50 之间的商品价格减 5 元,库存量再新增 300 件。
2. 在 sh_goods 表中查询评分小于 4 的商品的不同分类 id。

第6章 多表操作

学习目标

- 掌握多表之间的内连接、左外连接以及右外连接查询
- 掌握子查询的分类以及带关键字的子查询
- 熟悉外键约束的添加、删除以及关联表之间的操作

前面章节所涉及的都是针对一张表的操作，即单表操作。然而实际开发中业务逻辑较为复杂，通常都需要对两张以上的表进行操作，即多表操作。本章将针对多表操作进行详细讲解。

6.1 多表查询

6.1.1 联合查询

联合查询是多表查询的一种方式，在保证多个 SELECT 语句的查询字段数相同的情况下，合并多个查询的结果。联合查询经常应用在分表操作中，具体会在后面的章节中详细讲解，此处读者只需掌握联合查询的作用及语法即可。联合查询的基本语法格式如下。

```
SELECT…
UNION [ALL | DISTINCT] SELECT…
[UNION [ALL | DISTINCT] SELECT…];
```

在上述语法中，UNION 是实现联合查询的关键字，ALL 和 DISTINCT 是联合查询的选项。其中，ALL 表示保留所有的查询结果；DISTINCT 是默认值，可以省略，表示去除完全重复的记录。

例如，在 shop.sh_goods 表中，以联合查询的方式获取 category_id 为 9 的商品 id、name 和 price，以及 category_id 为 6 的商品 id、name 和 keyword。具体 SQL 语句如下。

```
mysql> USE shop;
Database changed
mysql> SELECT id, name, price FROM sh_goods WHERE category_id = 9
    -> UNION
    -> SELECT id, name, keyword FROM sh_goods WHERE category_id = 6;
```

```
+----+---------+---------+
| id | name    | price   |
+----+---------+---------+
|  7 | 头戴耳机 | 109.00  |
|  5 | 智能手机 | 电子产品 |
+----+---------+---------+
2 rows in set (0.00 sec)
```

在上述 SQL 语句中，SELECT 查询的字段个数必须相同，且联合查询的结果中只保留第一个 SELECT 语句对应的字段名称，即使 UNION 后 SELECT 查询的字段与第一个 SELECT 查询的字段表达含义或数据类型不同，MySQL 也仅会根据查询字段出现的顺序，对结果进行合并。例如，category_id 为 9 的 price 字段和 category_id 为 6 的 keyword 字段，在合并时只保留了 price 字段的名称，而 keyword 字段的值依然合并到了 price 字段下。

除此之外，若要对联合查询的记录进行排序等操作，需要使用圆括号"()"包裹每一个 SELECT 语句，在 SELECT 语句内或在联合查询的最后添加 ORDER BY 语句。并且若要排序生效，必须在 ORDER BY 后添加 LIMIT 限定联合查询排序的数量，通常推荐使用大于表记录数的任意值。

例如，以联合查询的方式，对 sh_goods 表中 category_id 为 3 的商品按价格升序排序，其他类型的产品按价格降序排序，查询的商品信息为 id、name 和 price。具体 SQL 语句及执行结果如下。

```
mysql> (SELECT id, name, price FROM sh_goods WHERE category_id <> 3
    -> ORDER by price DESC LIMIT 7)
    -> UNION
    -> (SELECT id, name, price FROM sh_goods WHERE category_id = 3
    -> ORDER by price ASC LIMIT 3);
+----+------------+---------+
| id | name       | price   |
+----+------------+---------+
|  4 | 超薄笔记本  | 5999.00 |
|  8 | 办公电脑    | 2000.00 |
|  5 | 智能手机    | 1999.00 |
|  9 | 收腰风衣    |  299.00 |
|  7 | 头戴耳机    |  109.00 |
|  6 | 桌面音箱    |   69.00 |
| 10 | 薄毛衣      |   48.00 |
|  1 | 2B 铅笔     |    0.50 |
|  3 | 碳素笔      |    1.00 |
|  2 | 钢笔        |   15.00 |
+----+------------+---------+
10 rows in set (0.00 sec)
```

在上述 SQL 语句中，在每个联合查询的 SELECT 语句中添加 ORDER BY 和 LIMIT，让每个查询语句按照指定的方式进行升序或降序排序。从执行结果中可知，前 7 条记录按照价格降序排序，后 3 条记录按价格升序排序。

6.1.2　连接查询

在实际应用中，可以根据多表之间存在的关联关系，将多张数据表连到一起。MySQL 中常用的连接查询有交叉连接、内连接、左外连接和右外连接。接下来将针对不同的连接查询进行详细讲解。

1. 交叉连接

交叉连接返回的结果是被连接的两个表中所有数据行的笛卡儿积。例如，商品分类表中有 3 个字段、4 条记录，商品表中有 5 个字段、10 条商品信息，那么交叉连接后的笛卡儿积就等于 4 * 10 条记录数，每条记录中含有 3+5 个字段。在 MySQL 中，交叉连接的基本语法格式如下。

```
SELECT 查询字段 FROM 表 1 CROSS JOIN 表 2;
```

上述语法格式中，CROSS JOIN 用于连接两个要查询的表，通过该语句可以查询两个表中所有的数据组合。

为了方便读者理解，下面以交叉连接商品分类表 sh_goods_category 和商品表 sh_goods 为例进行演示。具体 SQL 语句如下。

```
mysql> SELECT c.id cid, c.name cname, g.id gid, g.name gname
    -> FROM sh_goods_category AS c
    -> CROSS JOIN sh_goods AS g;
+-----+---------+-----+------------+
| cid | cname   | gid | gname      |
+-----+---------+-----+------------+
|  1  | 办公    |  1  | 2B 铅笔    |
|  1  | 办公    |  2  | 钢笔       |
|  1  | 办公    |  3  | 碳素笔     |
|  1  | 办公    |  4  | 超薄笔记本 |
|  1  | 办公    |  5  | 智能手机   |
|  1  | 办公    |  6  | 桌面音箱   |
|  1  | 办公    |  7  | 头戴耳机   |
|  1  | 办公    |  8  | 办公电脑   |
|  1  | 办公    |  9  | 收腰风衣   |
|  1  | 办公    | 10  | 薄毛衣     |
：因篇幅有限，此处省略了其余的查询结果
+-----+---------+-----+------------+
160 rows in set (0.00 sec)
```

在上述 SQL 语句中，使用 sh_goods_category 表中的每一条记录与 sh_goods 表中的记录进行连接，因此最后查询出的记录数为 160(sh_goods_category 的记录数 16 乘以 sh_goods 表中的记录数 10)。

值得一提的是，以上的交叉连接查询与 MySQL 中多表查询的语法等价。例如，上述示例可以使用以下 SQL 语句实现。

```
SELECT c.id, c.name, g.id, g.name
FROM sh_goods_category AS c, sh_goods AS g;
```

需要注意的是，交叉连接产生的结果是笛卡儿积，并没有实际应用的意义。

2. 内连接

内连接是一种常见的连接查询，它根据匹配条件返回第1个表与第2个表所有匹配成功的记录。在MySQL中，内连接的基本语法格式如下。

```
SELECT 查询字段 FROM 表 1
[INNER] JOIN 表 2 ON 匹配条件;
```

在上述语法中，ON用于指定内连接的查询条件，在不设置ON时，与交叉连接等价。此时可以使用WHERE完成条件的限定，效果与ON一样。但由于WHERE是限定已全部查询出来的记录，那么在数据量很大的情况下，此操作会浪费很多性能，所以此处推荐使用ON实现内连接的条件匹配。

为了方便读者理解，下面以内连接的方式查询商品表sh_goods和商品分类表sh_goods_category表中对应商品的分类id及name。具体SQL语句及执行结果如下。

```
mysql> SELECT g.id gid, g.name gname, c.id cid, c.name cname
    -> FROM sh_goods g JOIN sh_goods_category c
    -> ON g.category_id = c.id;
+-----+------------+-----+---------+
| gid | gname      | cid | cname   |
+-----+------------+-----+---------+
|   1 | 2B 铅笔    |   3 | 文具    |
|   2 | 钢笔       |   3 | 文具    |
|   3 | 碳素笔     |   3 | 文具    |
|   4 | 超薄笔记本 |  12 | 笔记本  |
|   5 | 智能手机   |   6 | 手机    |
|   6 | 桌面音箱   |   8 | 音箱    |
|   7 | 头戴耳机   |   9 | 耳机    |
|   8 | 办公电脑   |  10 | 电脑    |
|   9 | 收腰风衣   |  15 | 风衣    |
|  10 | 薄毛衣     |  16 | 毛衣    |
+-----+------------+-----+---------+
10 rows in set (0.00 sec)
```

从上述执行结果可知，只有sh_goods表的category_id与sh_goods_category表中的id相等的商品信息(gid和gname)和分类信息(cid和cname)才会被显示。另外，在查询数据时，由于sh_goods和sh_goods_category表中含有同名的字段，为了避免重名出现错误，使用“数据表.字段名”或“表别名.字段名”的方式进行区分。

除此之外，自连接查询是内连接中的一种特殊查询。它是指相互连接的表在物理上为同一个表，但逻辑上分为两个表。例如，要查询“钢笔”所在的分类下有哪些商品，就可以使用自连接查询。具体SQL语句及执行结果如下。

```
mysql> SELECT DISTINCT g1.id, g1.name FROM sh_goods g1
    -> JOIN sh_goods g2
    -> ON g2.name = '钢笔' AND g2.category_id = g1.category_id;
+----+--------+
| id | name   |
+----+--------+
|  1 | 2B 铅笔 |
|  2 | 钢笔    |
|  3 | 碳素笔  |
+----+--------+
3 rows in set (0.00 sec)
```

在上述 SQL 语句中，别名为 g1 和 g2 的表在物理上是同一个数据表 sh_goods，然后在 ON 的匹配条件中，指定 g1 表与 g2 表内商品名为“钢笔”且它们必须是同分类的记录进行自连接，从而获取 sh_goods 表中“钢笔”分类下的所有商品。

小提示： 在标准的 SQL 中，交叉连接（CROSS JOIN）与内连接（INNER JOIN）表示的含义不同，前者一般只连接表的笛卡儿积，而后者则是获取符合 ON 筛选条件的连接数据。但是在 MySQL 中，CROSS JOIN 与 INNER JOIN（或 JOIN）语法的功能相同，都可以使用 ON 设置连接的筛选条件，可以互换使用，但是此处不推荐读者将交叉连接与内连接混用。

3. 左外连接

左外连接是外连接查询中的一种，也可以将其称为左连接。它用于返回连接关键字（LEFT JOIN）左表中所有的记录，以及右表中符合连接条件的记录。当左表的某行记录在右表中没有匹配的记录时，右表中相关的记录将设为 NULL。其基本语法格式如下。

```
SELECT 查询字段
FROM 表 1 LEFT [OUTER] JOIN 表 2 ON 匹配条件
```

在上述语法中，关键字 LEFT [OUTER] JOIN 左边的表（表 1）被称为左表，也可称为主表；关键字右边的表（表 2）被称为右表，也可称为从表。其中，OUTER 在查询时可以省略。

下面利用左连接查询，以 sh_goods 表为主表，sh_goods_category 表为从表，查询所有 5 星商品对应的分类名称，具体 SQL 语句及执行结果如下。

```
mysql> SELECT g.id gid, g.name gname, c.id cid, c.name cname
    -> FROM sh_goods g LEFT JOIN sh_goods_category c
    -> ON g.category_id = c.id AND g.score = 5;
+-----+-----------+------+-------+
| gid | gname     | cid  | cname |
+-----+-----------+------+-------+
|   1 | 2B 铅笔    | NULL | NULL  |
|   2 | 钢笔       | NULL | NULL  |
```

```
|    3 | 碳素笔       | 3     | 文具  |
|    4 | 超薄笔记本   | NULL  | NULL  |
|    5 | 智能手机     | 6     | 手机  |
|    6 | 桌面音箱     | NULL  | NULL  |
|    7 | 头戴耳机     | NULL  | NULL  |
|    8 | 办公电脑     | NULL  | NULL  |
|    9 | 收腰风衣     | NULL  | NULL  |
|   10 | 薄毛衣       | NULL  | NULL  |
+----+------------+------+------+--
10 rows in set (0.00 sec)
```

从上述执行结果可知，左连接查询，即使主表 sh_goods 中的记录与从表 sh_goods_category 中任何记录都不符合匹配条件时，也会在查询结果中保留主表 sh_goods 中的此条记录（如 gid 等于 1 的记录），而从表 sh_goods_category 对应的字段值则为 NULL（如 cid、cname 字段）。

4. 右外连接

右外连接也是外连接查询中的一种，可以将其称为右连接。它用于返回连接关键字（RIGHT JOIN）右表（主表）中所有的记录，以及左表（从表）中符合连接条件的记录。当右表的某行记录在左表中没有匹配的记录时，左表中相关的记录将设为 NULL。其基本语法格式如下。

```
SELECT 查询字段
FROM 表 1 RIGHT [OUTER] JOIN 表 2 ON 匹配条件;
```

下面利用右连接查询，以 sh_goods_category 为主表，sh_goods 为从表，查询所有 5 星评价商品的对应分类名称，具体 SQL 语句及执行结果如下。

```
mysql> SELECT g.id gid, g.name gname, c.id cid, c.name cname
    -> FROM sh_goods g RIGHT JOIN sh_goods_category c
    -> ON c.id = g.category_id AND g.score = 5;
+------+----------+-----+----------+
| gid  | gname    | cid | cname    |
+------+----------+-----+----------+
|    3 | 碳素笔   |   3 | 文具     |
|    5 | 智能手机 |   6 | 手机     |
| NULL | NULL     |   1 | 办公     |
| NULL | NULL     |   2 | 耗材     |
| NULL | NULL     |   4 | 电子产品 |
| NULL | NULL     |   5 | 通讯     |
| NULL | NULL     |   7 | 影音     |
| NULL | NULL     |   8 | 音箱     |
| NULL | NULL     |   9 | 耳机     |
| NULL | NULL     |  10 | 电脑     |
```

```
| NULL    | NULL       |  11    | 台式电脑       |
| NULL    | NULL       |  12    | 笔记本         |
| NULL    | NULL       |  13    | 服装           |
| NULL    | NULL       |  14    | 女装           |
| NULL    | NULL       |  15    | 风衣           |
| NULL    | NULL       |  16    | 毛衣           |
+-------+--------+------ +----------+
16 rows in set (0.00 sec)
```

从上述执行结果可知,右连接查询的结果保存了主表 sh_goods_category 中的所有记录。其中,主表 sh_goods_category 中与从表 sh_goods 没有符合匹配条件的记录,从表 sh_goods 对应的字段 gid 和 gname 值为 NULL。

总之,外连接是最常用的一种查询数据的方式,分为左外连接(LEFT JOIN)和右外连接(RIGHT JOIN)。它与内连接的区别是,内连接只能获取符合连接条件的记录,而外连接不仅可以获取符合连接条件的记录,还可以保留主表与从表不能匹配的记录。

另外,右连接查询正好与左连接相反。因此,在应用外连接时仅调整关键字(LEFT 或 RIGHT JOIN)和主从表的位置,即可实现左连接和右连接的互换使用。

多学一招:USING 关键字

在连接查询时,若数据表连接的字段同名,则连接时的匹配条件可以使用 USING 代替 ON。其基本语法格式如下。

```
SELECT 查询字段
FROM 表 1 [CROSS|INNER|LEFT|RIGHT] JOIN 表 2
USING(同名的连接字段列表);
```

上述语法中,多个同名的连接字段之间使用逗号分隔。下面使用 USING 关键字实现自连接查询"钢笔"所在的分类下有哪些商品。具体 SQL 语句及执行结果如下。

```
mysql> SELECT DISTINCT g1.id, g1.name FROM sh_goods g1
    -> JOIN sh_goods g2
    -> USING(category_id) WHERE g2.name='钢笔';
+----+--------+
| id  | name      |
+----+--------+
|  1  | 2B 铅笔   |
|  2  | 钢笔      |
|  3  | 碳素笔    |
+----+--------+
3 rows in set (0.00 sec)
```

需要注意的是,USING 关键字在实际开发中并不常使用,原因在于设计表的时候不能确定使用相同的字段名称保存对应的数据。

6.2 子查询

6.2.1 什么是子查询

子查询可以理解为，在一个 SQL 语句 A(SELECT、INSERT、UPDATE 等)中嵌入一个查询语句 B,作为执行的条件或查询的数据源(代替 FROM 后的数据表),那么 B 就是子查询语句,它是一条完整的 SELECT 语句,能够独立地执行。

在含有子查询的语句中,子查询必须书写在圆括号内。SQL 语句首先会执行子查询中的语句,然后再将返回的结果作为外层 SQL 语句的过滤条件,当遇到同一个 SQL 语句中含有多层子查询时,它们执行的顺序是从最里层的子查询开始执行。

6.2.2 子查询分类

子查询的划分方式有多种,最常见的是以功能和位置进行划分。按子查询的功能可以分为标量子查询、列子查询、行子查询和表子查询;按子查询出现的位置可以分为 WHERE 子查询和 FROM 子查询。其中,标量子查询、列子查询和行子查询都属于 WHERE 子查询,而表子查询属于 FROM 子查询。

本节接下来以功能的划分方式为例详细讲解子查询的使用。

1. 标量子查询

所谓标量子查询指的是子查询返回的结果是一个数据,即一行一列。基本语法格式如下。

```
WHERE 条件判断 {=|<>}
(SELECT 字段名 FROM 数据源 [WHERE] [GROUP BY] [HAVING] [ORDER BY] [LIMIT]);
```

从上述语法可知,标量子查询利用比较运算符“=”或“<>”,判断子查询语句返回的数据是否与指定的条件相等或不等,然后根据比较结果完成相关需求的操作。其中,数据源表示一个符合二维表结构的数据,如数据表。

下面利用标量子查询的方式,从 sh_goods_category 表中获取商品名为“智能手机”的商品分类名称。具体 SQL 语句及执行结果如下。

```
mysql> SELECT name FROM sh_goods_category
    -> WHERE id = (SELECT category_id FROM sh_goods WHERE name='智能手机');
+-------+
| name  |
+-------+
| 手机  |
+-------+
1 row in set (0.00 sec)
```

从上述 SQL 语句可知,根据需求即可确定第一条 SELECT 语句的主要查询对象,如商品分类名称;根据提示的条件即可确定子查询语句要获取的数据,如获取“智能手机”对应的

category_id 分类 id；最后在主要查询语句中利用 WHERE 完成条件的判断，如 sh_goods_category 表中的 id 是否等于子查询返回的 category_id 值，若相等则返回分类名。

2. 列子查询

所谓列子查询指的是子查询返回的结果是一个字段符合条件的所有数据，即一列多行。基本语法格式如下。

```
WHERE 条件判断 {IN|NOT IN}
(SELECT 字段名 FROM 数据源 [WHERE] [GROUP BY] [HAVING] [ORDER BY] [LIMIT]);
```

从上述语法可知，列子查询利用比较运算函数"IN()"或"NOT IN()"，判断指定的条件是否在子查询语句返回的结果集中，然后根据比较结果完成相关需求的操作。

下面利用列子查询的方式，从 sh_goods_category 表中获取已添加了商品的商品分类名称。具体 SQL 语句及执行结果如下。

```
mysql> SELECT name FROM sh_goods_category
    -> WHERE id IN (SELECT DISTINT category_id FROM sh_goods);
+---------+
| name    |
+---------+
| 文具    |
| 笔记本  |
| 手机    |
| 音箱    |
| 耳机    |
| 电脑    |
| 风衣    |
| 毛衣    |
+---------+
8 rows in set (0.00 sec)
```

从上述 SQL 语句可知，首先确定要获取的主要查询对象，即商品分类名称；然后根据提示的条件获取 sh_goods 表中所有商品的 category_id 分类 id，接着在主查询对象中判断 sh_goods_category 中的 id 是否在子查询的结果中，若在则返回对应的分类名。

3. 行子查询

当子查询的结果是一条包含多个字段的记录（一行多列）时，称为行子查询。基本语法格式如下。

```
WHERE (指定字段名 1,指定字段名 2,…) =
(SELECT 字段名 1,字段名 2,… FROM 数据源 [WHERE] [GROUP BY]
[HAVING] [ORDER BY] [LIMIT]);
```

在上述语法中，行子查询返回的一条记录与指定的条件进行比较，比较的运算符通常使用"="，表示子查询的结果必须全部与指定的字段相等才满足 WHERE 指定的条件。除此之外，还可以使用 MySQL 中其他的比较运算符，如"<>"">"等，它们表示的含义如表 6-1

所示。

表 6-1　不同运算符的行比较

不同运算符的行比较	描　　述
(a, b) = (x, y)	表达的含义等价于 (a=x)AND(b=y)
(a, b) <=>(x, y)	表达的含义等价于(a<=>x)AND(b<=>y)
(a, b) <> (x, y)或 (a, b) != (x, y)	表达的含义等价于 (a<>x)OR (b<>y)
(a, b) > (x, y)	表达的含义等价于 (a>x) OR ((a=x) AND (b>y))
(a, b) >= (x, y)	表达的含义等价于 (a>x) OR ((a=x) AND (b>=y))
(a, b) < (x, y)	表达的含义等价于 (a<x) OR ((a=x) AND (b<y))
(a, b) <= (x, y)	表达的含义等价于 (a<x) OR ((a=x) AND (b<=y))

从表 6-1 可知，行在相等比较(=或<=>)时，各条件之间是与的逻辑关系；在不等比较(<>或!)时，各条件之间是或的逻辑关系；在进行其他方式比较时，各条件之间的逻辑关系包含两种情况。因此，读者在选取行子查询的比较运算符时，要根据实际需求慎重选择。

下面利用行子查询的方式，从 sh_goods 表中获取价格最高，且评分最低的商品信息(id、name、price、score、content)。具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, name, price, score, content FROM sh_goods WHERE
    -> (price, score) = (SELECT MAX(price), MIN(score) FROM sh_goods);
+----+------------+---------+-------+----------+
| id | name       | price   | score | content  |
+----+------------+---------+-------+----------+
|  4 | 超薄笔记本 | 5999.00 |  2.50 | 轻小便携 |
+----+------------+---------+-------+----------+
1 row in set (0.00 sec)
```

在上述 SQL 语句中，首先确定要获取的主要查询对象，即商品信息；然后根据提示的条件获取 sh_goods 表中最高的价格以及最低的商品评分，接着在主要查询对象中，利用 WHERE 完成条件的判断。

4. 表子查询

表子查询指的是子查询的返回结果用于 FROM 数据源，它是一个符合二维表结构的数据，可以是一行一列、一列多行、一行多列或多行多列。基本语法格式如下。

```
SELECT 字段列表 FROM (SELECT 语句) [AS] 别名
[WHERE] [GROUP BY] [HAVING] [ORDER BY] [LIMIT];
```

在上述语法中，FROM 后的数据源都是表名。因此，当数据源是子查询时必须为其设置别名，同时也是为了将查询结果作为一个表使用时，可以进行条件判断、分组、排序以及限量等操作。

下面利用表子查询的方式，从 sh_goods 表中获取每个商品分类下价格最高的商品信息(id、name、price、category_id)。具体 SQL 语句及执行结果如下。

```
mysql>SELECT a.id, a.name, a.price, a.category_id FROM sh_goods a,
    ->(SELECT category_id, MAX(price) max_price FROM sh_goods
    ->GROUP BY category_id) b
    ->WHERE a.category_id =b.category_id AND a.price =b.max_price;
+----+------------+---------+-------------+
| id | name       | price   | category_id |
+----+------------+---------+-------------+
|  2 | 钢笔       |   15.00 |           3 |
|  4 | 超薄笔记本 | 5999.00 |          12 |
|  5 | 智能手机   | 1999.00 |           6 |
|  6 | 桌面音箱   |   69.00 |           8 |
|  7 | 头戴耳机   |  109.00 |           9 |
|  8 | 办公电脑   | 2000.00 |          10 |
|  9 | 收腰风衣   |  299.00 |          15 |
| 10 | 薄毛衣     |   48.00 |          16 |
+----+------------+---------+-------------+
8 rows in set (0.00 sec)
```

在上述语句中，表子查询用于获取每个分类下商品最高的价格和对应分类 id，然后将此查询结果作为一个数据表 b 使用，判断 sh_goods 表中分类 id 与价格是否与表 b 中的分类 id 和价格都相等，若条件符合，就可以获取每个分类下价格最高的商品信息。

6.2.3　子查询关键字

在 WHERE 子查询中，不仅可以使用比较运算符，还可以使用 MySQL 提供的一些特定关键字，如前面讲解过的 IN。除此之外，常用的子查询关键字还有 EXISTS、ANY 和 ALL。下面将分别对这些关键字的使用进行详细讲解。

1. 带 EXISTS 关键字的子查询

带 EXISTS 关键字的子查询返回的结果只有 0 和 1 两个值。其中，0 代表不成立，1 代表成立。其基本语法格式如下。

```
WHERE EXISTS(子查询语句);
```

在上述语法中，EXISTS 关键字用于判断子查询语句是否有返回的结果，若存在则返回 1，否则返回 0。

例如，在 sh_goods_category 表中存在名称为“厨具”的分类时，将 sh_goods 表中 id 等于 5 的商品修改为电饭煲，价格修改为 599，分类修改为“厨具”对应的 id。具体 SQL 语句及执行结果如下。

```
mysql>UPDATE sh_goods SET name='电饭煲', price=599,
    ->category_id=(SELECT id FROM sh_goods_category WHERE name='厨具')
    ->WHERE EXISTS(SELECT id FROM sh_goods_category WHERE name='厨具')
    ->AND id=5;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0
```

在上述 SQL 语句中，sh_goods_category 表中不存在厨具时，子查询无结果，则 EXISTS() 的返回结果为 0。此时 UPDATE 语句的更新条件不满足，不会更新 sh_goods 中对应的语句。当需要相反的操作时，还可以使用 NOT EXISTS 判断子查询的结果，当没有返回结果时则返回 1，否则返回 0。

2. 带 ANY 关键字的子查询

在 SQL 语句中使用带 ANY 关键字的子查询时，表示给定的判断条件，只要符合 ANY 子查询结果中的任意一个，就返回 1，否则返回 0。基本语法格式如下。

```
WHERE 表达式 比较运算符 ANY(子查询语句);
```

在上述语法中，当比较运算符为"="时，其执行的效果等价于 IN 关键字。例如，从 sh_goods_category 中获取分类下商品价格小于 500 的商品分类名称。具体 SQL 语句及执行结果如下。

```
mysql> SELECT name FROM sh_goods_category
    -> WHERE id =
    -> ANY(SELECT DISTINCT category_id FROM sh_goods WHERE price<500);
+-------+
| name  |
+-------+
| 文具  |
| 音箱  |
| 耳机  |
| 风衣  |
| 毛衣  |
+-------+
5 rows in set (0.00 sec)
```

在上述 SQL 语句中，首先获取 ANY 子查询语句的结果，共 5 条 cid 的记录，分别为 3、8、9、15、16；然后将 sh_goods_category 表中的 id 与子查询结果集进行比较，只要 id 与子查询结果集中的一个值相等，就返回该条记录对应的 name 值，因此输出的结果为文具、音箱、耳机、风衣和毛衣。

若将上述 SQL 语句中的"="换为"<>"，就会获取 sh_goods_category 表中全部的商品分类名称。具体 SQL 语句及执行结果如下。

```
mysql> SELECT name FROM sh_goods_category
    -> WHERE id <>
    -> ANY(SELECT DISTINCT category_id FROM sh_goods WHERE price<500);
+----------+
| name     |
+----------+
| 办公     |
| 耗材     |
| 文具     |
| 电子产品 |
```

```
| 通讯       |
| 手机       |
| 影音       |
| 音箱       |
| 耳机       |
| 电脑       |
| 台式电脑   |
| 笔记本     |
| 服装       |
| 女装       |
| 风衣       |
| 毛衣       |
+----------+
16 rows in set (0.00 sec)
```

在上述 SQL 语句中，sh_goods_category 表的 id 值分别为 1 到 16，ANY 子查询的结果为 3、8、9、15、16。因此，sh_goods_category 表的 id 值只要与 ANY 子查询结果中的一个不等，就表示匹配成功，因此最后查询的结果为 sh_goods_category 表中全部分类名。

注意： MySQL 中还有一个关键字 SOME，与 ANY 的功能完全相同。MySQL 在设计时添加 SOME 的原因在于，英文语法中虽然 SOME 和 ANY 的语法含义相同，但是 NOT ANY 和 NOT SOME 的含义区别很大。前者表示一点也不，相当于 NOT ALL，而后者仅用于否定部分内容。因此，为了便于以英文为母语的开发者理解，设计出了带 SOME 和 ANY 关键字的子查询。

3. 带 ALL 关键字的子查询

在 SQL 语句中使用带 ALL 关键字的子查询时，表示给定的判断条件只有全部符合 ALL 子查询的结果时，才返回 1，否则返回 0。基本语法格式如下。

```
WHERE 表达式 比较运算符 ALL(子查询语句);
```

例如，获取 sh_goods 表中分类 id 为 3 下的商品价格比分类为 8 下的商品价格都低的商品信息（id、name、price、keyword）。具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, name, price, keyword FROM sh_goods
    -> WHERE category_id=3 AND price <
    -> ALL(SELECT DISTINCT price FROM sh_goods WHERE category_id=8);
+----+--------+-------+----------+
| id | name   | price | keyword  |
+----+--------+-------+----------+
|  1 | 2B铅笔 |  0.50 | 文具     |
|  2 | 钢笔   | 15.00 | 文具     |
|  3 | 碳素笔 |  1.00 | 文具     |
+----+--------+-------+----------+
3 rows in set (0.00 sec)
```

在上述 SQL 语句中，首先通过 ALL 子查询语句获取分类 id 等于 8 的所有商品价格，然后获取分类等于 3 且价格比子查询的所有结果都小的商品信息。

值得一提的是，带 ANY、SOME 或 ALL 关键字的子查询，不能使用“<=>”比较运算符。另外，若子查询结果与条件匹配时有 NULL，那么此条记录不参与匹配。

6.3 外键约束

在数据库设计时，为了保证不同表中相同含义数据的一致性和完整性，可为数据表添加外键约束。例如，员工表中添加了部门表中不存在的部门 id，此时就会出现数据信息保存不对等的情况，若在员工表中将部门 id 设置为外键，即可对相关的操作产生约束。例如，员工表只能插入部门表中含有的记录 id。接下来针对外键约束的使用进行详细讲解。

6.3.1 添加外键约束

外键指的是在一个表中引用另一个表中的一列或多列，被引用的列应该具有主键约束或唯一性约束，从而保证数据的一致性和完整性。其中，被引用的表称为主表；引用外键的表称为从表。

1. 添加外键约束

要想在创建数据表(CREATE TABLE)或修改数据结构(ALTER TABLE)时添加外键约束，在相应的位置添加以下 SQL 语句即可，其基本语法如下所示：

```
[CONSTRAINT symbol] FOREIGN KEY [index_name] (index_col_name,…)
REFERENCES tbl_name (index_col_name,…)
[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT}]
[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT}]
```

在上述语法中，MySQL 可以通过 FOREIGN KEY…REFERENCES 关键字向数据表中添加外键约束。其中，可选关键字 CONSTRAINT 用于定义外键约束的名称 symbol，如果省略，MySQL 将会自动生成一个名字。index_name 也是可选参数，表示外键索引名称，如果省略，MySQL 也会在建立外键时自动创建一个外键索引，加快查询速度。

语法中第一行的参数 “index_col_name，…”表示从表中外键名称列表；“tbl_name”表示主表，主表后的参数列表“index_col_name，…”表示主键约束或唯一性约束字段。“ON DELETE”与“ON UPDATE”用于设置主表中的数据被删除或修改时，从表对应数据的处理办法，其后的各参数具体说明如表 6-2 所示。

表 6-2 添加外键约束的参数说明

参数名称	功能描述
RESTRICT	默认值。拒绝主表删除或修改外键关联字段
CASCADE	主表中删除或更新记录时，同时自动删除或更新从表中对应的记录
SET NULL	主表中删除或更新记录时，使用 NULL 值替换从表中对应的记录(不适用于 NOT NULL 字段)
NO ACTION	与默认值 RESTRICT 相同，拒绝主表删除或修改外键关联字段
SET DEFAULT	设默认值，但 InnoDB 目前不支持

需要注意的是，目前只有 InnoDB 存储引擎支持外键约束。且建立外键关系的两个数据表的相关字段数据类型必须相似，也就是要求字段的数据类型可以相互转换。例如，INT 和 TINYINT 类型的字段可以建立外键关系，而 INT 和 CHAR 类型的字段则不可以建立外键关系。

为了读者更好地理解，下面在 mydb 数据库中，以员工表 employees 和部门表 department 为例，讲解如何在 CREATE TABLE 和 ALTER TABLE 时添加外键约束的两种方式。

（1）CREATE TABLE 时添加外键约束。

为从表创建外键约束时，首先要保证数据库中已存在主表，否则程序会报“不能添加外键约束”的错误。具体 SQL 语句及执行结果如下。

```
#① 在 mydb 数据库下创建主表
mysql>CREATE TABLE mydb.department (
    ->    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '部门编号',
    ->    name VARCHAR(50) NOT NULL COMMENT '部门名称'
    ->) DEFAULT CHARSET=utf8;
Query OK, 0 rows affected (0.01 sec)
#② 在 mydb 数据库下创建从表，添加外键约束
mysql>CREATE TABLE mydb.employees (
    ->id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '员工编号',
    ->name VARCHAR(120) NOT NULL COMMENT '员工姓名',
    ->dept_id INT UNSIGNED NOT NULL COMMENT '商品分类编号',
    ->CONSTRAINT FK_ID FOREIGN KEY(dept_id) REFERENCES department(id)
    ->ON DELETE RESTRICT ON UPDATE CASCADE
    ->) DEFAULT CHARSET=utf8;
Query OK, 0 rows affected (0.01 sec)
```

在上述 SQL 语句中，为 employees 表中的 dept_id 字段添加外键约束，与主表 department 中的主键 id 相关联。同时，利用 ON DELETE 指定从表此关联字段含有数据时，拒绝主表 department 执行删除操作，利用 ON UPDATE 设置主表 department 执行更新操作时，从表 employees 中的相关字段也执行更新操作。

值得一提的是，定义外键约束名称（如 FK_ID）时，不能加单引号和双引号。例如，添加外键约束时，使用 CONSTRAINT 'FK_ID' 或 CONSTRAINT "FK_ID" 的设置方式都会报错。

（2）ALTER TABLE 时添加外键约束。

对于已经创建的数据表，则可以通过 ALTER TABLE 的方式添加外键约束。例如，若 mydb 数据库中已有两个数据表 department 和 employees，employees 表在创建时未添加外键约束，此时就可以通过以下的 ALTER TABLE 方式实现。

```
ALTER TABLE mydb.employees
ADD CONSTRAINT FK_ID FOREIGN KEY(dept_id) REFERENCES department(id)
ON DELETE RESTRICT ON UPDATE CASCADE;
```

从以上示例可知，外键约束在 CREATE TABLE 和 ALTER TABLE 语法中添加的位置不同，但是基本实现语法全部相同。

2. 查看外键约束

在添加完外键约束后，可以利用 DESC 查看数据表 employees 中添加了外键约束的字段信息，具体 SQL 语句及执行结果如下。

```
mysql> DESC mydb.employees dept_id;
+----------+------------------+------+-----+---------+-------+
| Field    | Type             | Null | Key | Default | Extra |
+----------+------------------+------+-----+---------+-------+
| dept_id  | int(10) unsigned | NO   | MUL | NULL    |       |
+----------+------------------+------+-----+---------+-------+
1 row in set (0.00 sec)
```

从以上的执行结果可知，添加了外键约束的 dept_id 字段的 Key(索引)值为 MUL，表示非唯一性索引(MULTIPLE KEY)，值可以重复。由此可见，在创建外键约束时，MySQL 会自动为没有索引的外键字段创建索引。

另外，读者还可以在 MySQL 中使用 SHOW CREATE TABLE 查看 employees 表的详细结构。具体 SQL 语句及执行结果如下。

```
mysql> SHOW CREATE TABLE mydb.employees\G
*************** 1. row ***************
       Table: employees
Create Table: CREATE TABLE `employees` (
    `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '员工编号',
    `name` varchar(120) NOT NULL COMMENT '员工姓名',
    `dept_id` int(10) unsigned NOT NULL COMMENT '商品分类编号',
    PRIMARY KEY (`id`),
  KEY `FK_ID` (`dept_id`),
  CONSTRAINT `FK_ID` FOREIGN KEY (`dept_id`) REFERENCES `department` (`id`) ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

在上述结果中，为 dept_id 字段添加外键约束后，当 dept_id 没有索引时，服务器就会自动为其创建与外键同名的索引。数据表的默认存储引擎为 InnoDB，关于存储引擎与索引会在后面的章节详细讲解，此处了解即可。

值得一提的是，由于 ON DELETE RESTRICT 设置的拒绝主表的删除操作属于默认值，因此显示表创建语句时省略了此设置。若设置为删除主表记录，则从表对应字段设置为 NULL，即可在表的详细结构中显示到 ON DELETE SET NULL 的设置。

6.3.2 关联表操作

从前面的学习可知，实体之间具有一对一、一对多和多对多的联系，而具有关联的表中的数据，可以通过连接查询的方式获取，并且在没有添加外键约束时，关联表中的数据插入、更新和删除操作互不影响。但是对于添加了外键约束的关联表而言，数据的插入、更新和删

除操作就会受到一定的约束。接下来对具有外键约束关系的关联表操作进行详细讲解。

1. 添加数据

一个具有外键约束的从表在插入数据时，外键字段的值会受主表数据的约束，保证从表插入的数据必须符合约束规范的要求。如从表外键字段不能插入主表中不存在的数据。

例如，主表 department 中未添加数据时，向 employees 表中插入一条记录（名称为 Tom 的用户所属部门编号为 3）。具体 SQL 语句及执行结果如下。

```
mysql> INSERT INTO mydb.employees (name, dept_id) VALUES ('Tom', 3);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`mydb`.`employees`,
CONSTRAINT `FK_ID` FOREIGN KEY (`dept_id`) REFERENCES `department` (`id`) ON UPDATE CASCADE)
```

从以上的执行结果可知，从表外键字段插入的值必须选取主表中相关联字段已经存在的数据，否则就会报以上的错误提示信息。

下面为 department 表添加一条 id 值为 3，分类名为“研发部”的记录，然后再利用上述示例中的 SQL 语句为 employees 表添加数据。具体 SQL 语句及执行结果如下。

```
mysql> INSERT INTO mydb.department (id, name) VALUES (3, '研发部');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO mydb.employees (name, dept_id) VALUES ('Tom', 3);
Query OK, 1 row affected (0.00 sec)
```

从上述执行结果可知，在主表 department 中含有 id 为 3 的部门信息后，从表 employees 中才能插入此部门的用户信息。

2. 更新数据

对于建立外键约束的关联数据表来说，若对主表进行更新操作，从表将按照其建立外键约束时设置的 ON UPDATE 参数自动执行相应的操作。例如，当参数设置为 CASCADE 时，如果主表发生更新，则从表也会对相应的字段进行更新。

下面对具有外键约束关系的 employees（从表）和 department（主表）进行操作，具体 SQL 语句及执行结果如下。

```
mysql> UPDATE mydb.department SET id = 1 WHERE name = '研发部';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> SELECT name, dept_id FROM mydb.employees;
+------+---------+
| name | dept_id |
+------+---------+
| Tom  |       1 |
+------+---------+
1 row in set (0.00 sec)
```

从以上的执行结果可知，仅将主表 department 中名为“研发部”的 id 修改为 1 后，从表 employees 中的相关用户（Tom）的外键 dept_id 也同时被修改为 1。

小提示：外键约束在使用时既有一定的优势，同时又会带来一定的问题，具体如下。

（1）优势。

- 外键可节省开发量。
- 外键能约束数据有效性，防止非法数据的插入。

（2）劣势。

- 使用外键约束，会带来额外的开销。
- 主表被锁定时，会引发从表也被锁定。
- 删除主表的数据时，需先删除从表的数据。
- 含有外键约束的从表字段不能修改表结构。

其中，关于锁的内容会在后面的章节中详细讲解，此处了解即可。

3. **删除数据**

同样地，对于已建立外键约束的关联数据表来说，若要对主表执行删除操作，从表将按照其建立外键约束时设置的ON DELETE参数自动执行相应的操作。例如，当参数设置为RESTRICT时，如果主表进行删除操作，同时从表中的外键字段有关联记录，就会阻止主表的删除操作。

下面对具有外键约束关系的employees（从表）和department（主表）进行操作，具体SQL语句及执行结果如下。

```
mysql> DELETE FROM mydb.department WHERE id = 1;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`mydb`.`employees`,
CONSTRAINT `FK_ID` FOREIGN KEY (`dept_id`) REFERENCES `department` (`id`) ON UPDATE CASCADE)
```

从上述执行结果可知，在删除主表department中id等于1的记录时，由于从表employees中含有部门编号等于1的用户信息，程序就会给出以上的错误提示信息。

此时，若要删除具有ON DELETE RESTRICT约束关系的主表记录时，一定要先删除从表中对应的数据，然后再删除主表中的数据。具体SQL语句及执行结果如下所示。

```
mysql> DELETE FROM mydb.employees WHERE dept_id = 1;
Query OK, 1 row affected (0.00 sec)
mysql> DELETE FROM mydb.department WHERE id = 1;
Query OK, 1 row affected (0.00 sec)
```

从上述的操作可知，关联表在删除操作时使用DISTRICT严格模式，主表中每条记录的删除，都要保证从表中没有相关记录的对应数据，这会对开发造成很大的不便。因此，对于添加外键约束的ON DELETE一般都使用SET NULL模式，即删除主表记录时，将从表中对应的记录设置为NULL，同时要保证从表中对应的外键字段允许为空，否则不允许设置该模式。

6.3.3 删除外键约束

在实际开发中，根据业务逻辑的需求，需要解除两个表之间的关联关系时，就要删除外键约束。基本语法格式如下。

```
ALTER TABLE 表名 DROP FOREIGN KEY 外键名;
```

下面以解除员工表 employees 与部门表 department 之间的外键约束为例进行演示，具体 SQL 语句及执行结果如下。

```
mysql>ALTER TABLE mydb.employees DROP FOREIGN KEY FK_ID;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

在删除 employees 表的外键约束后，下面利用 DESC 查询 employees 表中删除了外键约束的字段信息，具体 SQL 语句及执行结果如下。

```
mysql>DESC mydb.employees dept_id;
+----------+------------------+------+-----+---------+-------+
| Field    | Type             | Null | Key | Default | Extra |
+----------+------------------+------+-----+---------+-------+
| dept_id  | int(10) unsigned |  NO  | MUL | NULL    |       |
+----------+------------------+------+-----+---------+-------+
1 row in set (0.00 sec)
```

从上述执行结果可知，在删除 dept_id 的外键约束后，dept_id 的 Key 值依然为 MUL，这是由于删除外键约束时并不会自动删除系统创建的普通索引，此时可以通过 SHOW CREATE TABLE 查看。具体 SQL 语句及执行结果如下。

```
mysql>SHOW CREATE TABLE mydb.employees \G
*************** 1. row ***************
       Table: employees
Create Table: CREATE TABLE `employees` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '员工编号',
  `name` varchar(120) NOT NULL COMMENT '员工姓名',
  `dept_id` int(10) unsigned NOT NULL COMMENT '商品分类编号',
  PRIMARY KEY (`id`),
  KEY `FK_ID` (`dept_id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

若要在删除外键约束后，同时删除系统为外键创建的普通索引，则需要通过手动删除索引的方式完成，具体 SQL 语句及执行结果如下。

```
mysql>ALTER TABLE mydb.employees DROP KEY FK_ID;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

完成上述操作后，读者可再次利用 DESC 查看已删除的外键字段的 Key 值为空，SHOW CREATE TABLE 时表中的外键约束以及普通索引全部已删除。

6.4 动手实践：多表查询练习

数据库的学习在于多看、多学、多想、多动手，只有将理论与实际相结合，才能够体现出

数据开发与管理的重要性,展现知识学习的价值与力量。接下来请结合本章所学的知识完成多表查询练习。

【实践目标】

此实践的目标就是能够根据文字提示,完成 shop 数据库中的 sh_goods_attr 商品属性表和 sh_goods_attr_value 商品属性值表的各种查询需求。

【实践需求】

(1) 查询 id 为 6 的分类所具有的属性信息,将属性按照层级并排显示。
(2) 查询 id 为 5 的商品的所有属性信息,将属性名称和属性值并排显示。
(3) 查询 id 为 1 的属性的所有子属性值。
(4) 查询拥有属性值个数大于 1 的商品 id 和名称。

【动手实践】

(1) 查询 id 为 6 的分类(手机)所具有的属性信息,将属性按照层级并排显示。

在查询时,应注意 sh_goods_attr 表中有一个 sort 字段,表示排序值,在输出属性信息时,应按照 sort 字段的值升序排列。实现该需求的 SQL 语句和执行结果如下。

```
mysql> SELECT a.sort sort1, a.name name1, b.sort sort2, b.name name2
    -> FROM sh_goods_attr a
    -> JOIN sh_goods_attr b ON a.id = b.parent_id
    -> WHERE a.category_id = 6
    -> ORDER BY a.sort ASC, b.sort ASC;
+-------+----------+-------+------------+
| sort1 | name1    | sort2 | name2      |
+-------+----------+-------+------------+
|     0 | 基本信息 |     0 | 机身颜色   |
|     0 | 基本信息 |     1 | 输入方式   |
|     0 | 基本信息 |     2 | 操作系统   |
|     1 | 屏幕     |     0 | 屏幕尺寸   |
|     1 | 屏幕     |     1 | 屏幕材质   |
|     1 | 屏幕     |     2 | 分辨率     |
|     2 | 摄像头   |     0 | 前置摄像头 |
|     2 | 摄像头   |     1 | 后置摄像头 |
|     3 | 电池信息 |     0 | 电池容量   |
|     3 | 电池信息 |     1 | 是否可拆卸 |
+-------+----------+-------+------------+
10 rows in set (0.00 sec)
```

(2) 查询 id 为 5 的商品(智能手机)的所有属性信息,将属性名称和属性值并排显示。实现该需求的具体 SQL 语句和执行结果如下。

```
mysql> SELECT b.name, a.attr_value FROM sh_goods_attr_value a
    -> JOIN sh_goods_attr b ON a.attr_id = b.id
    -> WHERE a.goods_id = 5;
+------------+------------+
| name       | attr_value |
+------------+------------+
| 机身颜色   | 黑色       |
| 输入方式   | 触摸屏     |
| 操作系统   | Android    |
| 屏幕尺寸   | 5.5寸      |
| 屏幕材质   | IPS        |
| 分辨率     | 1920*1080  |
| 前置摄像头 | 1600万     |
| 后置摄像头 | 800万      |
| 电池容量   | 3500mAh    |
| 是否可拆卸 | 否         |
+------------+------------+
10 rows in set (0.00 sec)
```

在上述结果中,属性名称只有子级属性,若需要显示父级属性名称并按照 sort 排序值排列,则使用如下 SQL 语句来实现。

```
mysql> SELECT c.sort sort1, c.name name1, b.sort sort2, b.name name2,
    -> a.attr_value FROM sh_goods_attr_value a
    -> JOIN sh_goods_attr b ON a.attr_id = b.id
    -> JOIN sh_goods_attr c ON b.parent_id = c.id
    -> WHERE a.goods_id = 5
    -> ORDER BY c.sort ASC, b.sort ASC;
+-------+----------+-------+------------+------------+
| sort1 | name1    | sort2 | name2      | attr_value |
+-------+----------+-------+------------+------------+
|     0 | 基本信息 |     0 | 机身颜色   | 黑色       |
|     0 | 基本信息 |     1 | 输入方式   | 触摸屏     |
|     0 | 基本信息 |     2 | 操作系统   | Android    |
|     1 | 屏幕     |     0 | 屏幕尺寸   | 5.5寸      |
|     1 | 屏幕     |     1 | 屏幕材质   | IPS        |
|     1 | 屏幕     |     2 | 分辨率     | 1920*1080  |
|     2 | 摄像头   |     0 | 前置摄像头 | 1600万     |
|     2 | 摄像头   |     1 | 后置摄像头 | 800万      |
|     3 | 电池信息 |     0 | 电池容量   | 3500mAh    |
|     3 | 电池信息 |     1 | 是否可拆卸 | 否         |
+-------+----------+-------+------------+------------+
10 rows in set (0.00 sec)
```

(3) 查询 id 为 1 的属性的所有子属性值,有两种实现方式,一种是通过连接表查询实现,另一种是通过子查询实现。下面演示利用子查询来实现,具体 SQL 语句如下。

```
mysql> SELECT attr_value FROM sh_goods_attr_value WHERE attr_id IN
    -> (SELECT id FROM sh_goods_attr WHERE parent_id = 1);
+------------+
| attr_value |
+------------+
| 黑色       |
| 触摸屏     |
| Android    |
+------------+
3 rows in set (0.00 sec)
```

（4）查询拥有属性值个数大于 1 的商品 id 和名称。首先通过子查询在商品属性值表中找出所有属性值个数大于 1 的商品 id，然后获取这些商品的 id 和名称。具体 SQL 语句和执行结果如下。

```
mysql> SELECT id, name FROM sh_goods WHERE id IN
    -> (SELECT goods_id FROM sh_goods_attr_value GROUP BY goods_id
    -> HAVING COUNT(id) > 1);
+----+----------+
| id | name     |
+----+----------+
|  5 | 智能手机 |
+----+----------+
1 row in set (0.00 sec)
```

6.5 本章小结

本章主要讲解了多表操作的相关知识，包括子查询、多表之间的关联关系、关联表建立外键约束，以及关联表的增删改查操作。通过本章的学习，希望大家能够熟练掌握多表查询中的连接查询和子查询。

6.6 课后练习

一、填空题

1. 带有________关键字的子查询只要查询结果中有一个符合要求就返回真。
2. 子查询根据位置的不同可分为 WHERE 子查询和________。
3. 含有 5 个字段的商品表（3 条记录）与含有 2 个字段的分类表（4 条记录）交叉连接查询后记录的总数为________。
4. ________查询在不设置连接条件时与交叉连接等价。
5. 在行子查询中表达式(a, b) <> (x, y)等价于含有逻辑运算符的________。

二、判断题

1. 在表结构中，含有外键约束的表称为主表。（　　）

2. WHERE 的指定条件只要符合 SOME 子查询结果中的任何一个都表示成立。(　　)

3. 目前只有 InnoDB 引擎类型支持外键约束。(　　)

4. 默认情况下,联合查询会去除完全重复的记录。(　　)

5. 建立外键约束的主表和从表数据类型必须完全相同。(　　)

三、选择题

1. 下列不属于 WHERE 子查询的是(　　)。
   A. 标量子查询　B. 列子查询　C. 行子查询　D. 表子查询

2. 添加外键约束时,设置(　　)可同步更新主表和从表对应的记录。
   A. ON UPDATE RESTRICT　B. ON UPDATE CASCADE
   C. ON UPDATE SET NULL　D. 以上答案都不正确

3. 以下连接查询中,(　　)仅会保留符合条件的记录。
   A. 左外连接　B. 右外连接　C. 内连接　D. 自连接

4. 下列选项中数据 FROM 子查询的是(　　)。
   A. EXISTS 子查询　B. 表子查询
   C. 行子查询　D. 以上答案都不正确

5. 下面关于外键约束的说法错误的是(　　)。
   A. 只有 InnoDB 存储引擎的数据表才支持外键约束
   B. 默认情况下,主表记录修改的同时修改从表的记录
   C. 从表外键字段插入的值必须选取主表中相关联字段已经存在的数据
   D. 默认情况下,从表含有关联记录则拒绝删除主表记录

四、实训题

1. 结合本章出现的所有数据表,获取含有基本信息属性的商品 id 和 name。

2. 结合本章出现的所有数据表,查询属性不小于两个的商品信息(编号和名称)。

第7章 用户与权限

学习目标

- 了解用户与权限的作用
- 掌握 CREATE USE 创建用户
- 掌握 ALTER USE 设置密码
- 掌握 GRANT 授予用户权限

在前面的章节中,都是通过 root(超级用户)登录数据库进行相关的操作。在正常的工作环境中,为了保证数据库的安全,数据库的管理员会对需要操作数据库的人员分配用户名、密码以及可操作的权限范围,让其仅能在自己权限范围内操作。本章针对 MySQL 中的用户与权限进行详细讲解。

7.1 用户与权限概述

用户是数据库的使用者和管理者,MySQL 通过用户的设置来控制数据库操作人员的访问与操作范围。在安装 MySQL 时,系统会自动安装一个名为 mysql 的数据库,该数据库主要用于维护数据库的用户以及权限的控制和管理。其中,MySQL 中的所有用户信息都保存在 mysql. user 数据表中。

使用 DESC 即可查看 user 表含有的 45 个字段,为了方便读者学习,下面以表格的形式列举 user 表中的一些常用字段,具体如表 7-1 所示。

表 7-1 user 表常用字段

字段名	数据类型	默认值
Host	char(60)	
User	char(32)	
Select_priv	enum('N', 'Y')	N
Insert_priv	enum('N', 'Y')	N
Update_priv	enum('N', 'Y')	N
Delete_priv	enum('N', 'Y')	N

续表

字　段　名	数 据 类 型	默　认　值
Create_priv	enum('N', 'Y')	N
Drop_priv	enum('N', 'Y')	N
Reload_priv	enum('N', 'Y')	N
Shutdown_priv	enum('N', 'Y')	N
Process_priv	enum('N', 'Y')	N
File_priv	enum('N', 'Y')	N
Grant_priv	enum('N', 'Y')	N
ssl_type	enum('','ANY','X509','SPECIFIED')	
ssl_cipher	blob	NULL
x509_issuer	blob	NULL
x509_subject	blob	NULL
max_questions	int(11) unsigned	0
max_updates	int(11) unsigned	0
max_connections	int(11) unsigned	0
max_user_connections	int(11) unsigned	0
plugin	char(64)	mysql_native_password
authentication_string	text	NULL
password_expired	enum('N','Y')	N
password_last_changed	timestamp	NULL
password_lifetime	smallint(5) unsigned	NULL
account_locked	enum('N','Y')	N

在表 7-1 中，根据字段的功能可将其分为 6 类，分别为客户端访问服务器的账号字段、验证用户身份的字段、安全连接的字段、资源限制的字段、权限字段以及账户是否锁定的字段。为了方便读者理解，下面分别对这 6 类字段进行详细讲解。

1. 账号字段

Host 和 User 字段共同组成的复合主键用于区分 MySQL 中的账户，User 字段用于代表用户的名称，Host 字段表示允许访问的客户端 IP 地址或主机地址，当 Host 的值为“*”时，表示所有客户端的用户都可以访问。

下面通过 SELECT 查询 user 表中默认用户的 Host 和 User 值，具体 SQL 语句及执行结果如下。

```
mysql> SELECT host, user FROM mysql.user;
+-----------+---------------+
| host      | user          |
+-----------+---------------+
| localhost | mysql.session |
| localhost | mysql.sys     |
| localhost | root          |
+-----------+---------------+
3 rows in set (0.00 sec)
```

从上述执行结果可知，在 user 表中，除了默认的 root 超级用户外，MySQL 5.7 中还额外地新增了两个用户 mysql. session 和 mysql. sys。前者用于用户身份验证，后者用于系统模式对象的定义，防止 DBA(数据库管理员)重命名或删除 root 用户时发生错误。

默认情况下，用户 mysql. session 和 mysql. sys 已被锁定，使得数据库操作人员无法使用这两个用户通过客户端连接 MySQL 服务器。因此，建议读者不要随意解锁和使用 mysql. session 和 mysql. sys 用户，否则可能会有意想不到的事情发生。

2. 身份验证字段

在 MySQL 5.7 中，mysql. user 表中已不再包含 Password 字段，而是使用 plugin 和 authentication_string 字段保存用户身份验证的信息。其中，plugin 字段用于指定用户的验证插件名称，authentication_string 字段是根据 plugin 指定的插件算法对账户明文密码(如 123456)加密后的字符串。

下面通过 SELECT 查询 user 表中 root 用户默认的 plugin 和 authentication_string 值，具体 SQL 语句及执行结果如下。

```
mysql> SELECT plugin, authentication_string FROM mysql.user
    -> WHERE user='root';
+-----------------------+-------------------------------------------+
| plugin                | authentication_string                     |
+-----------------------+-------------------------------------------+
| mysql_native_password | *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |
+-----------------------+-------------------------------------------+
1 row in set (0.00 sec)
```

从以上的执行结果可知，MySQL 中 root 用户的默认验证插件名为 mysql_native_password。而 authentication_string 字段保存的则是一串不能看出具体含义的值，相对于能够直接看懂的明文密码(如 123456)，它是经过加密处理的暗码。

除此之外，与身份验证的账号密码相关的字段还有 password_expired(密码是否过期)、password_last_changed(密码最后一次修改的时间)以及 password_lifetime(密码的有效期)。

3. 安全连接字段

在客户端与 MySQL 服务器连接时，除了可以基于账户名以及密码的常规验证外，还可以判断当前连接是否符合 SSL 安全协议，与其相关的字段有以下几种。

（1）ssl_type：用于保存安全连接的类型，它的可选值有"（空）、ANY（任意类型）、X509（X509 证书）、SPECIFIED（规定的）4 种。

（2）ssl_cipher：用于保存安全加密连接的特定密码。

（3）x509_issuer：保存由 CA 签发的有效的 X509 证书。

（4）x509_subject：保存包含主题的有效的 X509 证书。

需要注意的是，通常标准的发行版 MySQL 默认未启用 SSL 加密连接，读者可以通过以下 SQL 语句查看。

```
mysql> SHOW VARIABLES LIKE 'have_openssl';
+---------------+----------+
| Variable_name | Value    |
+---------------+----------+
| have_openssl  | DISABLED |
+---------------+----------+
1 row in set, 1 warning (0.00 sec)
```

在上述执行结果中，have_openssl 的取值为 DISABLED，表示已禁用加密。因此，读者在本章仅需了解用户在连接服务器时可以选择 SSL 加密安全连接即可。

4. 资源限制字段

在 mysql.user 表中提供的以"max_"开头的字段，保存对用户可使用的服务器资源的限制，用来防止用户登录 MySQL 服务器后的不法或不合规范的操作浪费服务器的资源，各字段的具体含义如下所示。

（1）max_questions：保存每小时允许用户执行查询操作的最多次数。

（2）max_updates：保存每小时允许用户执行更新操作的最多次数。

（3）max_connections：保存每小时允许用户建立连接的最多次数。

（4）max_user_connections：保存允许单个用户同时建立连接的最多数量。

以上列举的用户资源限制字段默认值均为 0，表示对此用户没有任何的资源限制。

5. 权限字段

在 mysql.user 表中提供的以"_priv"结尾的字段一共有 29 个，这些字段保存了用户的全局权限，如 Select_priv 查询权限、Insert_priv 插入权限、Update_priv 更新权限等。

其中，user 表对应的权限字段的数据类型都是 ENUM 枚举类型，取值只有 N 或 Y 两种。N 表示该用户没有对应权限，Y 表示该用户有对应权限。为了保证数据库的安全，这些字段的默认值都为 N，如果需要可以对其进行修改。

6. 账户锁定字段

在 mysql.user 表中提供的 account_locked 字段用于保存当前用户是锁定还是解锁状态。该字段是一个枚举类型，当其值为 N 时表示解锁，此用户可以用于连接服务器；当其值为 Y 时表示该用户已被锁定，不能用于连接服务器使用。

7.2 用户管理

7.2.1 创建用户

由于 MySQL 中所有用户的信息都保存在 mysql. user 表中，因此，创建用户可以直接利用 root 用户登录 MySQL 服务器后，向 mysql. user 表中插入记录，但是在开发中为保证数据的安全，并不推荐使用此方式创建用户。而是采用 MySQL 提供的 CREATE USER 和 GRANT 语句创建用户。其中，GRANT 语句在创建用户的同时还可以完成权限的设置，具体内容将在下一个小节中讲解。

使用 CREATE USER 语句每创建一个新用户，都会在 mysql. user 表中添加一条记录，同时服务器会自动修改相应的授权表。但需要注意的是，该语句创建的新用户默认情况下没有任何权限，需要使用 GRANT 进行授权。基本语法格式如下。

```
CREATE USER [IF NOT EXISTS]
账户名 [用户身份验证选项][, 账户名 [用户身份验证选项]]...
[WITH 资源控制选项][密码管理选项 | 账户锁定选项]
```

在上述语法中，CREATE USER 可以一次创建多个用户，多个用户之间使用逗号分隔。其中，账户名是由“用户名@主机地址”组成的，其余选项在创建用户时，若未设置则使用默认值，具体如表 7-2 所示。

表 7-2 CREATE USER 语句的选项

选 项	默 认 值
用户身份验证选项	由 default_authentication_plugin 系统变量定义的插件进行身份验证
加密连接协议选项	NONE
资源控制选项	N(表示无限制)
密码管理选项	PASSWORD EXPIRE DEFAULT
用户锁定选项	ACCOUNT UNLOCK

在表 7-2 中，用户身份验证选项的设置仅适用于其前面的用户名，可将其理解为某个用户的私有属性；其余的选项对声明中的所有用户都有效，可以将其理解为全局属性。

小提示： 创建用户时，用户名的设置不能超过 32 个字符，且区分大小写，但是主机地址不区分大小写。

为了读者更好地理解与使用，下面通过示例的方式一一演示用户的创建。

（1）创建最简单的用户。在创建用户时，若不指定主机地址、密码以及相关的用户选项，则表示此用户在访问 MySQL 服务器时，不限定客户端，不需要密码并且没有任何限制。具体 SQL 语句及执行结果如下。

```
mysql> CREATE USER 'test1';
Query OK, 0 rows affected (0.00 sec)
```

接下来，使用 SELECT 查看 mysql 数据库下的 user 表，查看创建的用户是否已添加到 user 表中保存。具体 SQL 语句及执行结果如下。

```
mysql> SELECT host, user FROM mysql.user;
+-----------+---------------+
| host      | user          |
+-----------+---------------+
| %         | test1         |
| localhost | mysql.session |
| localhost | mysql.sys     |
| localhost | root          |
+-----------+---------------+
4 rows in set (0.00 sec)
```

在上述执行结果中，host 的值为“%”时表示当前的用户可以在任何主机中连接 MySQL 服务器；当其值为“localhost”时，表示当前的用户只能从本地主机连接 MySQL 服务器。另外，host 的值为空字符串('')同样也可表示匹配所有客户端。

注意：用户名和主机名在设置时，若不包含空格、“-”等特殊字符，则可以省略引号。另外，当创建的用户名称为空字符串('')时，表示创建的是一个匿名用户，即登录 MySQL 服务器时不需要输入用户名和密码，这种操作会给 MySQL 服务器带来极大的安全隐患，因此不推荐用户创建并使用匿名用户操作 MySQL 服务器。

(2) 创建含有密码的用户。若要在创建用户的同时完成用户密码的设置，则可以选取用户身份验证选项的 IDENTIFIED BY 关键字。具体 SQL 语句及执行结果如下。

```
mysql> CREATE USER 'test2'@'localhost' IDENTIFIED BY '123456';
Query OK, 0 rows affected (0.00 sec)
```

在上述 SQL 语句中，IDENTIFIED BY 后指定的是字符串形式的明文密码。接着，通过 SELECT 查询存储在 user 表中 test2 用户的密码。具体 SQL 语句及执行结果如下。

```
mysql> SELECT plugin, authentication_string FROM mysql.user
    -> WHERE user= 'test2';
+-----------------------+-------------------------------------------+
| plugin                | authentication_string                     |
+-----------------------+-------------------------------------------+
| mysql_native_password | *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |
+-----------------------+-------------------------------------------+
1 row in set (0.00 sec)
```

从上述运行可知，创建用户时设置的明文密码“123456”，在 user 表中以默认的算法插件 mysql_native_password 将其转换为暗码。

除此之外，在设置用户密码时还可以指定对密码加密的插件，只需将“IDENTIFIED BY '明文密码'”修改为指定的选项即可，具体如表 7-3 所示。

表 7-3 用户身份验证选项

选 项	描 述
IDENTIFIED WITH 验证插件名	使用指定的身份验证插件对空凭证(未设置用户密码)进行加密
IDENTIFIED WITH 验证插件名 BY '明文密码'	利用指定的身份验证插件对明文密码进行加密
IDENTIFIED WITH 验证插件名 AS '哈希字符串'	指定身份验证的插件,并存储哈希加密字符串
IDENTIFIED BY PASSWORD '哈希字符串'	身份验证插件为默认,并存储哈希加密字符串

下面利用“IDENTIFIED WITH 验证插件名 BY '明文密码'”修改以上创建 test2 用户的 SQL 语句,具体如下。

```
CREATE USER 'test2'@'localhost'
IDENTIFIED WITH 'mysql_native_password' BY '123456';
```

在上述 SQL 语句中,IDENTIFIED WITH 关键字指明验证密码的插件名,关键字 BY 后跟明文的密码。通过上述设置,明文密码“123456”将采用指定的插件算法,存储在 user 表中的对应字段中。

(3) 同时创建多个用户。MySQL 中创建用户时,还可以一次创建多个,具体 SQL 语句及执行结果如下。

```
mysql>CREATE USER
    ->'test3'@'localhost' IDENTIFIED BY '333333',
    ->'test4'@'localhost' IDENTIFIED BY '444444';
Query OK, 0 rows affected (0.01 sec)
```

在上述语法中,在一个 CREATE USER 语句中可同时创建多个用户,多个用户之间使用逗号分隔,并且在创建每个用户时可以单独为其设置密码,省略用户身份验证选项时,表明此用户在登录服务器时可以免密登录,但为了保证数据安全,不推荐用户这样做。

(4) 设置用户可操作资源范围。在创建用户时,可以添加 WITH 直接为用户指定可操作的资源范围,如登录的用户在一小时内可以查询数据的次数等,具体的参数选项如表 7-4 所示。

表 7-4 可操作资源范围的参数选项

选 项	描 述
MAX_QUERIES_PER_HOUR	在任何一个小时内,允许此用户执行多少次查询
MAX_UPDATES_PER_HOUR	在任何一个小时内,允许此用户执行多少次更新
MAX_CONNECTIONS_PER_HOUR	在任何一个小时内,允许此用户执行多少次服务器连接
MAX_USER_CONNECTIONS	限制用户同时连接服务器的最大数量

在表 7-4 中,MAX_USER_CONNECTIONS 选项的值为 0 时,服务器将根据 max_user_connections 系统变量的值确定用户的同时连接数,若此变量值也为 0,表示对该用户没有限制。另外 MAX_QUERIES_PER_HOUR 选项不会计算从缓存中查询数据的次数。

例如，创建一个名为 test5 的本地用户，限制其每小时最多可以更新 10 次。具体 SQL 语句及执行结果如下。

```
mysql>CREATE USER
    ->'test5'@'localhost' IDENTIFIED BY '555555'
    ->WITH MAX_UPDATES_PER_HOUR 10;
Query OK, 0 rows affected (0.00 sec)
```

创建完成后，可以查看 user 表中的 max_updates 字段，具体 SQL 语句及执行结果如下。

```
mysql>SELECT max_updates FROM user WHERE user='test5';
+-------------+
| max_updates |
+-------------+
|          10 |
+-------------+
1 row in set (0.00 sec)
```

（5）设置有密码期限的用户。在创建用户时，不仅可以为用户设置密码，还可以为密码设置有效时间，具体参数如表 7-5 所示。

表 7-5　密码管理选项

选　　项	描　　述
PASSWORD EXPIRE	将密码标记为过期
PASSWORD EXPIRE DEFAULT	根据 default_password_lifetime 系统变量的指定设置密码的有效性
PASSWORD EXPIRE NEVER	密码永不过期
PASSWORD EXPIRE INTERVAL n DAY	将账户密码生存期设置为 n 天

在表 7-5 中，PASSWORD EXPIRE 选项表示在创建用户时含有此选项的用户在登录后，执行任何 SQL 语句操作前，都需要重置用户的密码，否则会给出一个“在执行此语句之前，必须使用 ALTER USER 语句重置密码”的错误提示信息。

注意：*在重置用户密码时，操作的用户必须要有全局的 CREATE USER 权限或 mysql 数据库的 UPDATE 特权。*

为了让读者更好地理解，下面创建一个名为 test6 的用户，并设置其密码每 180 天更改一次。具体 SQL 语句及执行结果如下。

```
mysql>CREATE USER 'test6'@'localhost' IDENTIFIED BY '666666'
    ->PASSWORD EXPIRE INTERVAL 180 DAY;
Query OK, 0 rows affected (0.00 sec)
```

值得一提的是，为了确保 MySQL 客户端用户本身的安全，通常情况下推荐每 3～6 个月变更一次数据库用户密码。

（6）设置用户是否锁定。利用 ACCOUNT 关键字可以为创建的用户设置是否被锁定，

它有两个可选值，分别为 LOCK（锁定）和 UNLOCK（解锁）。需要注意的是，被锁定的用户不能在客户端登录 MySQL 服务器。

下面创建一个锁定的用户 test 7。具体 SQL 语句及执行结果如下。

```
mysql>CREATE USER 'test7'@'localhost' IDENTIFIED BY '777777'
    ->PASSWORD EXPIRE ACCOUNT LOCK;
Query OK, 0 rows affected (0.00 sec)
```

此时，利用 SELECT 查看 mysql.user 表中名为 test7 的 account_ locked 字段，若其值为 Y，则表示当前创建的用户已被锁定。具体 SQL 语句及执行结果如下。

```
mysql>SELECT account_locked FROM mysql.user WHERE user='test7';
+----------------+
| account_locked |
+----------------+
| Y              |
+----------------+
1 row in set (0.00 sec)
```

此时若利用 test7 在客户端登录 MySQL 服务器，会报"账户被锁定，拒绝访问 test7@localhost"用户的提示信息。

7.2.2 设置密码

在对 MySQL 中的用户进行管理时，除了创建用户的同时设置密码外，还可为没有密码的用户、密码过期的用户或为指定用户修改密码。其基本语法格式如下。

```
#① 第 1 种语法
ALTER USER 账户名 IDENTIFIED BY '明文密码';
#② 第 2 种语法
SET PASSWORD [FOR 账户名] ='明文密码';
#③ 第 3 种语法
SET PASSWORD [FOR 账户名] =PASSWORD('明文密码);
```

在以上 3 种语法中，ALTER USER 是更改密码的首选 SQL 语句，推荐使用。第 2 种语法可能会被记录到服务器的日志或客户端的历史文件中，会有密码泄露的风险，因此建议用户尽量少地使用此方式设置密码。而第 3 种语法从 MySQL 5.7.6 开始已被弃用，并且将在未来的 MySQL 版本中被删除，因此本书中不推荐使用此语法。

为了读者更好地理解，下面以案例的方式讲解常见设置密码的方式。具体如下。

（1）为指定用户设置密码。mysql.user 表中的 test1 用户是一个可在任何主机中连接 MySQL 服务器的用户，但此用户在创建时未设置密码，下面将 test1 的密码设置为 123456，具体 SQL 语句如下。

```
mysql>ALTER USER 'test1'@'%' IDENTIFIED BY '123456';
Query OK, 0 rows affected (0.00 sec)
```

（2）为登录用户设置密码。若当前通过客户端连接 MySQL 服务器的用户是非匿名用

户，则可以使用 USER()函数更改自己的密码，而无须直接为自己的账户命名。

例如，修改当前正在连接 MySQL 服务器的 root 用户密码，将其设置为 000000。具体 SQL 语句及执行结果如下。

```
mysql>ALTER USER USER() IDENTIFIED BY '000000';
Query OK, 0 rows affected (0.00 sec)
```

值得一提的是，USER()函数获取的是客户端提供的用户和主机地址，它可能与当前通过 MySQL 服务器验证的用户与主机名不同，此时可以利用 MySQL 提供 CURRENT_USER()函数获取，具体 SQL 语句及执行结果如下。

```
mysql>SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| root@localhost |
+----------------+
1 row in set (0.00 sec)
```

多学一招：mysqladmin 修改用户密码

在 MySQL 的安装目录 bin 下还有一个 mysqladmin.exe 应用程序，它通常用于执行一些管理性的工作，以及显示服务器状态等。同时，在 MySQL 中也可以使用该命令修改用户的密码。基本语法格式如下。

```
mysqladmin -u 用户名 [-h 主机地址] -p password 新密码
```

在上述语法格式中，-u 用于指定待要修改的用户名，通常情况下指的是 root 用户；-h 用于指定对应的主机地址，省略时默认值为 localhost；-p 后面的 password 为关键字，而不是待修改用户的原密码。

例如，在命令行窗口中，使用 mysqladmin 命令，将 root 用户的密码修改为 123456，具体 SQL 语句及执行结果如下。

```
c:\mysql5.7\bin>mysqladmin -u root -p password 123456
Enter password: ******
mysqladmin: [Warning] Using a password on the command line interface can be insecure.
Warning: Since password will be sent to server in plain text, use ssl connection to ensure password
safety.
```

在上述命令行中，“-p password”后指定的是 root 用户修改后的新密码；而“Enter password”提示输入的密码指的是 root 用户的旧密码，只有此密码输入正确，才能完成用户密码的修改。此时，读者可通过“mysql -uroot -p123456”成功连接 MySQL 服务器。

值得一提的是，通过 mysqladmin 完成用户密码的设置时，会有两个安全警告提示。因此，为了确保密码安全应以安全连接方式连接 MySQL 服务器。在开发中一般不推荐使用此种方式修改用户密码，此处读者将其作为知识扩展中需了解的内容即可。

多学一招：SET PASSWORD与PASSWORD()函数

在前面为用户设置密码的语法中可以看出，SET PASSWORD在设置密码时，一种是添加PASSWORD()函数，一种是不添加，两者的区别在于含有PASSWORD()函数的密码根据old_passwords系统变量值(0表示mysql_native_password插件，2表示sha256_password插件)对指定的明文密码加密并验证；而不含有PASSWORD()函数的SET PASSWORD语句是将明文密码字符串传递给用户的验证插件，并将插件返回的结果保存到mysql.user表的字段中。

多学一招：root密码丢失找回

如果忘记MySQL服务的root用户密码，就不能通过以上设置密码的方式找回或重置。此时可以在MySQL的配置文件my.ini中添加skip-grant-tables选项，然后重启MySQL服务后再次利用root用户登录，就可以跳过密码的输入直接登录MySQL服务，为root用户设置密码。但是此种方式存在非常大的安全风险。因此，建议读者要慎重使用。

7.2.3 修改用户

用户创建完成后，管理员可以通过MySQL提供的专门SQL语句修改用户的密码、身份验证的方式、资源限制、密码的属性，以及账户的锁定和解锁的状态。基本语法格式如下。

```
ALTER USER [IF EXISTS]
账户名 [用户身份验证选项][, 账户名 [用户身份验证选项]]...
[WITH 资源限制选项][密码管理选项 | 账户锁定选项]
```

在上述语法中，ALTER USER可以同时修改一个或多个用户，多个用户之间使用逗号(,)分隔，并且语法中选项的可选值与创建用户时的选项完全相同。对于每个修改的用户，都会更新其在mysql.user表中对应的字段值，而未修改的字段仍然保留它原来的值。

关于ALTER USER方式修改用户密码已在上一个小节中讲解，下面通过案例的方式演示剩余用户选项修改的操作。

(1) 修改验证插件。例如，将mysql.user表中名为test1的用户验证插件修改为sha256_password，密码修改为111111，并将密码设置为立刻过期。具体SQL语句及执行结果如下。

```
mysql>ALTER USER test1
    ->IDENTIFIED WITH sha256_password BY '111111'
    ->PASSWORD EXPIRE;
Query OK, 0 rows affected (0.01 sec)
```

执行完上述SQL语句后，下面查看名为test1且插件算法为sha256_password的authentication_string字段的值。具体SQL语句及执行结果如下。

```
mysql>SELECT authentication_string FROM mysql.user
    ->WHERE user='test1' AND plugin='sha256_password';
+------------------------------------------------------------------+
| authentication_string                                            |
```

```
+----------------------------------------------------------------------+
| $5$sT
hN??Q┘ *2Y\d$MqwcDkJk.Sfr8xm2rwNThCPjA9GppOn828jeT8UIBsD               |
+----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

从上述执行结果可知，用户密码的加密算法不同，则此记录对应的 authentication_string 字段保存的值也不相同。密码加密的结果不适合输出展示，但是它们都是用于验证用户身份的字符串。

（2）解锁用户。例如，解锁 mysql.user 表中被锁定的用户 test7。具体 SQL 语句及执行结果如下。

```
mysql>ALTER USER 'test7'@'localhost' ACCOUNT UNLOCK;
Query OK, 0 rows affected (0.00 sec)
```

完成解锁后，读者可利用此账户在客户端连接 MySQL 服务器，这里不再演示。

（3）同时修改多个用户的资源限定。例如，修改 mysql.user 表中的 test1 和 test2 用户，限定单个用户最多可同时建立两个连接，同时将 test1 的验证插件修改为 MySQL 的默认值，test2 用户的密码修改为 222222。具体 SQL 语句及执行结果如下。

```
mysql>ALTER USER
    ->'test1' IDENTIFIED WITH mysql_native_password,
    ->'test2'@'localhost' IDENTIFIED BY '222222'
    ->WITH max_user_connections 2;
Query OK, 0 rows affected (0.00 sec)
```

完成上述操作后，读者可打开 3 个客户端，选取 test1 和 test2 中的一个用户连接 MySQL 服务器，直到在第 3 个客户端连接时会给出警告和错误提示，表示当前用户仅允许同时建立两个连接。

多学一招：为用户重命名

在利用 ALTER USER 修改用户时只能修改指定账户的相关选项，如密码、验证插件、资源控制选项等，而不能够为用户重新命名，此时可以使用 MySQL 专门提供的 RENAME USER 语句实现，基本语法格式如下。

```
RENAME USER
旧用户名 1 TO 新用户名 1 [, 旧用户名 2 TO 新用户名 2] …
```

在上述语法中，RENAME USER 在为用户重命名时，旧用户名与新用户名之间使用 TO 关键字连接，同时为多个用户重命名时使用逗号（,）进行分隔。

下面将 mysql.user 表中名为 test6 的用户名修改为“xiaoming”，具体 SQL 语句及执行结果如下。

```
mysql>RENAME USER 'test6'@'localhost' TO 'xiaoming'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

按照以上的 SQL 语句执行完成后，读者可利用 SELECT 语句查看 mysql. user 表中用户名称的变化，这里不再演示。

需要注意的是，被重命名的旧用户不存在或新用户名已存在时，系统会报错。

7.2.4 删除用户

在 MySQL 中经常会创建多个普通用户管理数据库，但如果发现某些用户是没有必要的，就可以将其删除，通常删除用户的方式采用 MySQL 提供的专门 SQL 语句。其基本语法格式如下。

```
DROP USER [IF EXISTS] 账户名 [, 账户名] …;
```

在上述语法中，MySQL 5. x 版本之后 DROP USER 语句可以同时删除一个或多个 MySQL 中的指定用户，并会同时从授权表中删除账户对应的权限行。在 MySQL 5. x 之前的版本中，在删除用户前必须先回收用户的权限。其中，账户名与创建用户的格式相同，由"用户名@主机地址"组成。

为了读者更好地理解，下面以删除 mysql. user 中 test7 用户为例进行演示。具体 SQL 语句及执行结果如下。

```
mysql> DROP USER IF EXISTS test7;
Query OK, 0 rows affected, 1 warning (0.01 sec)
```

在上述示例中，在不添加 IF EXISTS 关键字时，若删除了一个不存在的用户，则该语句的执行会发生错误；在添加后，会在删除不存在的用户时生成一个警告作为提示。其中，在删除账户时，如果省略主机地址，则默认为"'%'"。

值得一提的是，当 DROP USER 语句删除当前正在打开的用户时，则该用户的会话不会被自动关闭。只有在该用户会话关闭后，删除操作才会生效，再次登录将会失败。另外，利用已删除的用户登录服务器创建的数据库或对象不会因此删除操作而失效。

7.3 权限管理

在实际项目开发中，为了保证数据的安全，数据库管理员需要为不同层级的操作人员分配不同的权限，限制登录 MySQL 服务器的用户只能在其权限范围内操作。同时管理员还可以根据不同的情况为用户增加权限或回收权限，从而控制数据操作人员的权限。本节将针对 MySQL 的权限管理进行详细讲解。

7.3.1 授予权限

MySQL 中的权限信息根据其作用范围，分别存储在 mysql 数据库中的不同数据表中。当 MySQL 启动时会自动加载这些权限信息，并将这些权限信息读取到内存中。与权限相关的数据表如表 7-6 所示。

表 7-6　与权限相关的数据表

数　据　表	描　　述
user	保存用户被授予的全局权限
db	保存用户被授予的数据库权限
tables_priv	保存用户被授予的表权限
columns_priv	保存用户被授予的列权限
procs_priv	保存用户被授予的存储过程权限
proxies_priv	保存用户被授予的代理权限

表 7-6 中保存权限的数据表，根据权限的操作内容可将权限大致分为数据权限、结构权限以及管理权限(通常只有管理员有管理权限)。接下来通过表 7-7 列举一下 MySQL 中可以授予和取消的权限。

表 7-7　MySQL 提供的常见权限

分类	权　　限	权 限 级 别	描　　述
数据权限	SELECT	全局、数据库、表、列	SELECT
	UPDATE	全局、数据库、表、列	UPDATE
	DELETE	全局、数据库、表	DELETE
	INSERT	全局、数据库、表、列	INSERT
	SHOW DATABASES	全局	SHOW DATABASES
	SHOW VIEW	全局、数据库、表	SHOW CREATE VIEW
	PROCESS	全局	SHOW PROCESSLIST
结构权限	DROP	全局、数据库、表	允许删除数据库、表和视图
	CREATE	全局、数据库、表	创建数据库、表
	CREATE ROUTINE	全局、数据库	创建存储过程
	CREATE TABLESPACE	全局	允许创建、修改或删除表空间和日志文件组
	CREATE TEMPORARY TABLES	全局、数据库	CREATE TEMPORARY TABLE
	CREATE VIEW	全局、数据库、表	允许创建或修改视图
	ALTER	全局、数据库、表	ALTER TABLE
	ALTER ROUTINE	全局、数据库、存储过程	允许删除或修改存储过程
	INDEX	全局、数据库、表	允许创建或删除索引
	TRIGGER	全局、数据库、表	允许触发器的所有操作
	REFERENCES	全局、数据库、表、列	允许创建外键

续表

分类	权　　限	权 限 级 别	描　　述
管理权限	SUPER	全局	允许使用其他管理操作，如 CHANGE MASTER TO 等
	CREATE USER	全局	CREATE USER、DROP USER、RENAME USER 和 REVOKEALL PRIVILEGES
	GRANT OPTION	全局、数据库、表、存储过程、代理	允许授予或删除用户权限
	RELOAD	全局	FLUSH 操作
	PROXY		与代理的用户权限相同
	REPLICATION CLIENT	全局	允许用户访问主服务器或从服务器
	REPLICATION SLAVE	全局	允许复制从服务器读取的主服务器二进制日志事件
	SHUTDOWN	全局	允许使用 mysqladmin shutdown
	LOCK TABLES	全局、数据库	允许在有 SELECT 表权限上使用 LOCK TABLES

在表 7-7 中，权限级别指的就是权限可以被应用在哪些数据库的内容中。例如，SELECT 权限可以被授予到全局(任意数据库下的任意内容)、数据库(指定数据库下的任意内容)、表(指定数据库下的指定数据表)、列(指定数据库下的指定数据表中的指定字段)。

MySQL 提供的 GRANT 用于实现用户权限的授予，其基本语法格式如下。

```
GRANT
权限类型 [字段列表][, 权限类型 [字段列表]] ...
ON [目标类型] 权限级别
TO 账户名 [用户身份验证选项] [, 账户名 [用户身份验证选项]] ...
[REQUIRE 连接方式]
[WITH {GRANT OPTION | 资源控制选项}]
```

在上述语法中，权限类型指的就是表 7-7 中的权限列，字段列表用于设置列权限，ON 后的目标类型默认为 TABLE，表示将全局、数据库、表或列中的某些权限授予给指定的用户。另外，其值还可以是 FUNCTION(函数)或 PROCEDURE(存储过程)，此处读者了解这两个名词即可，具体的介绍会在后面的章节中讲解。

权限级别用于定义全局权限、数据库权限和表权限，添加 GRANT OPTION 表示当前账户可以为其他账户进行授权。其余各参数均与 CREATE USER 中的用户选项相同，这里不再赘述。

为了读者更好地理解，下面首先使用 SHOW GRANTS [FOR 账户]查看一下 root 和 test1 用户被授权的情况，具体 SQL 语句及执行结果如下。

```
mysql> SHOW GRANTS FOR 'root'@'localhost';
+--------------------------------------------------------------+
| Grants for root@localhost                                    |
```

```
+-----------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION   |
|GRANT PROXY ON ''@'' TO 'root'@'localhost' WITH GRANT OPTION           |
+-----------------------------------------------------------------------+
2 rows in set (0.00 sec)
mysql>SHOW GRANTS FOR 'test1'@'%';
+-------------------------------------+
| Grants for test1@%                  |
+-------------------------------------+
| GRANT USAGE ON *.* TO 'test1'@'%'   |
+-------------------------------------+
1 row in set (0.00 sec)
```

在上述 SQL 语句中，若在客户端使用的是 root 用户登录 MySQL 服务器，则此处在查看 root 用户权限时，可以省略 FOR 'root'@'localhost'。

另外，查询结果中 ALL PRIVILEGES 表示除 GRANT OPTION（授权权限）和 PROXY（代理权限）外的所有权限，USAGE 表示没有任何权限。ON 后的"*.*"表示全局级别的权限，即 MySQL 服务器下的所有数据库下的所有表，"''@''"表示任何主机中的匿名用户。

在了解了 MySQL 中都有哪些权限以及 GRANT 语法后，接下来将以案例的形式分别演示如何使用 GRANT 为用户授权以及如何创建新用户。

（1）为用户授权。在为用户授予权限时，可以分为 6 个不同的级别，具体的实现在于 ON 子句以及权限列表的不同，如表 7-8 所示。

表 7-8　用户授权语法

权限级别	实现语法
全局权限	GRANT 权限列表 **ON** *.* TO 账户名[WITH GRANT OPTION];
数据库级权限	GRANT 权限列表 **ON 数据库名.** * TO 账户名[WITH GRANT OPTION];
表级权限	GRANT 权限列表 **ON 数据库名.表名** TO 账户名[WITH GRANT OPTION];
列级权限	GRANT **权限类型（字段列表）**[,…]**ON 数据库名.表名** TO 账户名[WITH GRANT OPTION];
存储过程权限	GRANT **EXECUTE**|**ALTER ROUTINE**|**CREATE ROUTINE** **ON** {[*.*|**数据库名.***]|**PROCEDURE 数据库名.存储过程**} TO 账户名 [WITH GRANT OPTION];
代理权限	GRANT **PROXY ON 账户名** TO 账户名 1 [, 账户名 2] …[WITH GRANT OPTION]

需要注意的是，要想使用 GRANT 语句为用户授权，必须要拥有 GRANT OPTION 权限；且在启用 read_only 系统变量时，还必须要拥有 SUPER 权限。

为了让读者更好地理解，下面以授予 test1 用户 shop.sh_goods 表的 SELECT 权限，以及对 name 和 price 字段的插入权限为例进行演示，具体 SQL 语句及执行结果如下。

```
mysql> GRANT SELECT, INSERT (name, price)
    -> ON shop.sh_goods
    -> TO 'test1'@'%';
Query OK, 0 rows affected (0.00 sec)
```

在上述 SQL 语句中，SELECT 权限是表级权限(shop. sh_goods 表)，INSERT 是列级权限(shop. sh_goods 表中的 name 和 price 字段)。因此，在 SQL 语句执行成功后，若要查看 test1 的权限，可以到 mysql. tables_priv 中查看表权限，到 mysql. columns_priv 中查看列权限。具体 SQL 语句及执行结果如下。

```
mysql> SELECT db,table_name,table_priv,column_priv
    -> FROM mysql.tables_priv WHERE user = 'test1';
+-------+-------------+-------------+-------------+
| db    | table_name  | table_priv  | column_priv |
+-------+-------------+-------------+-------------+
| shop  | sh_goods    | Select      | Insert      |
+-------+-------------+-------------+-------------+
1 row in set (0.00 sec)
mysql> SELECT db,table_name,column_name,column_priv
    -> FROM mysql.columns_priv WHERE user= 'test1';
+-------+-------------+-------------+-------------+
| db    | table_name  | column_name | column_priv |
+-------+-------------+-------------+-------------+
| shop  | sh_goods    | name        | Insert      |
| shop  | sh_goods    | price       | Insert      |
+-------+-------------+-------------+-------------+
2 rows in set (0.00 sec)
```

从上述的查询中可以看出，test1 用户对 shop 数据库下的 goods 表有查询权限，对 goods 表中的 name 和 price 字段有插入数据的权限。除此之外，读者也可以利用前面学习过的 SHOW GRANTS [FOR 账户]语句查看指定账户整体的权限情况。

(2) 创建用户。默认情况下，在 MySQL 5.7 中，当 GRANT 语句中指定的账户不存在时，系统不支持自动创建用户，它会报一个在 user 表中找不到用户的提示信息。例如，为 user 表中不存在的 test8 用户授予 SELECT 权限，具体 SQL 语句及执行结果如下。

```
mysql> GRANT SELECT ON *.* TO 'test8'@'';
ERROR 1133 (42000): Can't find any matching row in the user table
```

为了解决上述的问题，只需确保 MySQL 中 NO_AUTO_CREATE_USER 模式未开启，就可以利用 GRANT 自动创建一个不存在的用户。

接下来，可以使用 SET 清除 MySQL 中的默认 SQL 模式，然后再重新执行以上的 SQL 语句。具体如下。

```
#① 清空默认 SQL 模式
mysql> SET sql_mode = '';
Query OK, 0 rows affected, 1 warning (0.00 sec)
#② 授予权限时，创建不存在的用户
```

```
mysql>GRANT SELECT ON * .* TO 'test8'@'';
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

从上述执行结果可知,修改后 GRANT 可以新建用户,同时也可为新用户设置权限。但在执行结果中还有一条警告信息,具体内容如下。

```
mysql>SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1287
Message: Using GRANT for creating new user is deprecated and will be removed in future release. Create new user with CREATE USER statement.
1 row in set (0.00 sec)
```

从上可知,使用 GRANT 创建新用户的方式已被废弃,并在未来会被移出。因此,MySQL 官方推荐使用 CREATE USER 语句创建用户,使用 ALTER USER 语句修改用户的非特权选项(如验证插件、资源控制选项等),使用 GRANT 为新用户授予权限。所以,请读者在完成以上操作后,执行以下的操作,将 SQL 模式修改为 MySQL 默认的值。

```
mysql>SET sql_mode =@@global.sql_mode;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

7.3.2　回收权限

在 MySQL 中,为了保证数据库的安全,需要将用户不必要的权限回收。例如,数据管理员发现某个用户不应该具有 DELETE 权限,就应该及时将其收回。为此,MySQL 专门提供了一个 REVOKE 语句用于回收指定用户的权限。其基本语法格式如下。

```
REVOKE 权限类型 [(字段列表)] [, 权限类型[(字段列表)]] …
ON [目标类型] 权限级别 FROM 账户名 [, 账户名] …
```

在上述语法中,权限类型、目标类型以及权限级别与授予权限 GRANT 的参数相同。

下面以删除 test1 用户对 shop.sh_goods 表的 name 和 price 字段的插入权限为例进行演示,具体 SQL 语句及执行结果如下。

```
mysql>REVOKE INSERT (name, price)
    ->ON shop.sh_goods FROM 'test1'@'%';
Query OK, 0 rows affected (0.00 sec)
```

按照以上的操作执行运行成功后,下面使用 test1 用户登录 MySQL 服务器,向 shop.sh_goods 表中插入数据,此时会看到以下的提示信息。

```
mysql>INSERT INTO shop.sh_goods(name, price) VALUES('test', 23);
ERROR 1142 (42000): INSERT command denied to user 'test1'@'localhost' for table 'sh_goods'
```

从上可知,系统拒绝 test1 用户对 sh_goods 表执行插入操作。

另外,当用户的权限比较多,并且想要一次性将其全部收回时,使用上述语句就会比较

麻烦，为此，可以使用以下的方式回收权限，其基本语法格式如下。

```
#① 回收表7-7中的所有权限以及可为其他用户授权的权限
REVOKE ALL [PRIVILEGES], GRANT OPTION FROM 账户名 [, 账户名] …
#② 回收用户的代理权限
REVOKE PROXY ON 账户名 FROM 账户名1 [, 账户名2] …
```

在上述语法中，“ALL [PRIVILEGES]”中的PRIVILEGES在使用时可以省略，它表示除去GRANT OPTION(授予权限)和PROXY(代理权限)外的所有权限。

7.3.3 刷新权限

刷新权限指的是从系统数据库mysql中的权限表中重新加载用户的权限。原因在于GRANT、CREATE USER等操作会将服务器的缓存信息保存到内存中，而REVOKE、DROP USER操作并不会同步到内存中，因此可能会造成服务器内存的消耗，所以在REVOKE、DROP USER后推荐读者使用MySQL提供的FLUSH PRIVILEGES重新加载用户的权限。具体SQL语句及执行结果如下。

```
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
```

另外，刷新权限也可以利用msyqladmin命令完成，具体使用方式如下。

```
#方式1
mysqladmin  -uroot -p reload
#方式2
mysqladmin  -uroot -p flush-privileges
```

上述操作命令在执行时，首先会提示输入root用户的密码。其中，reload与flush-privileges都可以用于重载权限信息。

7.4 动手实践：用户与权限练习

数据库的学习在于多看、多学、多想、多动手，只有将理论与实际相结合，才能够体现出数据库开发与管理的重要性，展现知识学习的价值与力量。接下来请结合本章所学的知识完成与shop数据库同名的用户操作。

【实践目标】

此实践的目标就是能够根据文字提示，完成用户的创建、修改、密码的设置以及权限授予等操作。

【实践需求】

(1) 为shop数据库创建一个可在“127.0.0”C类网络中的任何主机访问的同名用户shop，初始密码为123456。

（2）为 shop 数据库的同名用户设置首次登录重置密码。
（3）将 shop 数据库的同名用户密码重置为 2c5-q8h。
（4）为 shop 数据库的同名用户授予查看 sh_goods 表信息的权限。
（5）回收与 shop 数据库同名用户对 sh_goods 表信息的查询权限。

【动手实践】

1. 创建用户

根据【实践需求】(1)可知要创建的用户名为 shop，可以访问的主机地址为“127.0.0.%”，初始密码为 123456。具体 SQL 语句及执行结果如下。

```
mysql> CREATE USER IF NOT EXISTS 'shop'@'127.0.0.%'
    -> IDENTIFIED BY '123456';
Query OK, 0 rows affected (0.01 sec)
```

2. 将密码设置为过期

根据【实践需求】(2)可知要实现 shop 用户首次登录重置密码，就需要使用 ALTER USER 将用户的密码期限设置为过期即可。具体 SQL 语句及执行结果如下。

```
mysql> ALTER USER 'shop'@'127.0.0.%' PASSWORD EXPIRE;
Query OK, 0 rows affected (0.00 sec)
```

3. 修改用户密码

根据【实践需求】(1)和(2)的操作，即可使用 shop 用户登录 MySQL 服务器，但是在登录后，执行任何命令，都会被提示重置密码。例如，查看当前服务器中的所有数据库，具体 SQL 语句及执行结果如下。

```
mysql> SHOW DATABASES;
ERROR 1820 (HY000): You must reset your password using ALTER USER statement before executing this statement.
```

若要解决以上的问题，此时需要具有全局 CREATE USER 权限或对 mysql 数据库的 UPDATE 权限的用户才能修改 shop 用户的密码。这里使用 root 用户根据【实践需求】(3)将密码修改为“2c5-q8h”，具体 SQL 语句及执行结果如下。

```
mysql> ALTER USER 'shop'@'127.0.0.%' IDENTIFIED BY '2c5-q8h';
Query OK, 0 rows affected (0.00 sec)
```

修改完成后，再次使用 shop 用户登录 MySQL 服务器，可知默认未授权的情况下，该用户只能查看到一个 information_schema 数据库，其余数据库均不可见，并且在未授权的情况下不能对该数据库中的数据表执行增、删、改、查等操作。

4. 为用户设置权限

这里使用 root 用户根据【实践需求】(4)为名为 shop 的用户授予查看 shop.sh_goods

表的权限，具体 SQL 语句及执行结果如下。

```
mysql> GRANT SELECT ON shop.sh_goods TO 'shop'@'127.0.0.%';
Query OK, 0 rows affected (0.00 sec)
```

修改完成后，再次使用 shop 用户登录 MySQL 服务器，即可查看到的数据库有两个，分别为 information_schema 和 shop。而 shop 用户仅能查看到 shop 数据库中 sh_goods 数据表，其他数据表均不可见；同时也仅能查看 sh_goods 表中的数据，不能对 sh_goods 表执行其他操作（如 DROP），否则系统会报相关的错误信息。如下所示。

```
mysql> DROP TABLE shop.sh_goods;
ERROR 1142 (42000): DROP command denied to user 'shop'@'localhost' for table 'sh_goods'
```

5. 回收用户权限

这里使用 root 用户根据【实践需求】(5)的要求回收 shop 用户对 shop. sh_goods 表的 SELECT 权限。具体 SQL 语句及执行结果如下。

```
mysql> REVOKE SELECT ON shop.sh_goods FROM 'shop'@'127.0.0.%';
Query OK, 0 rows affected (0.00 sec)
```

完成上述操作后，利用 shop 用户登录客户端，执行查询 shop. sh_goods 表的操作，会看到以下的提示信息。

```
ERROR 1142 (42000): SELECT command denied to user 'shop'@'localhost' for table 'sh_goods'
```

上述信息表示用户'shop'@'localhost'没有表 sh_goods 的查询权限。除此之外，读者也可使用删除用户的方式回收授予此用户的所有权限。具体 SQL 语句如下。

```
DROP USER IF EXISTS 'shop'@'localhost';
```

7.5 本章小结

本章主要讲解了 MySQL 中用户与权限的操作。通过本章的学习，希望读者掌握如何创建用户、为用户设置密码、修改用户可操作的资源以及用户权限的授予及回收。

7.6 课后练习

一、填空题

1. 函数________可获取通过 MySQL 服务器验证的账户。
2. MySQL 提供的________可用于刷新用户权限。
3. 在 mysql 数据库中，列权限的所有信息存储在________表中。
4. 在 mysql. user 表中________和________用于区分 MySQL 中的用户。

5. 语句________可以查看指定用户的权限。

二、判断题

1. GRANT 创建用户时使用 IF NOT EXISTS 可避免用户存在而出错。(　　)
2. 授权时,ALL PRIVILEGES 包括为其他用户授予权限的权限。(　　)
3. mysql.user 表用于保存全局级别的权限。(　　)
4. 'test'@'127.0.0.1'中的 IP 地址'127.0.0.1' 表示本地主机。(　　)
5. root 用户密码丢失后不能再找回,只能重新安装 MySQL。(　　)

三、选择题

1. 下列选项中可以重置用户密码的是(　　)。
   A. ALTER USER　　B. RENAME USER
   C. CREATE USER　　D. DROP UESR
2. 以下不属于 ALL PRIVILEGES 的权限是(　　)。
   A. PROXY　　B. SELECT
   C. CREATE USER　　D. DROP
3. 下列 mysql 数据库中用于保存用户名和密码的表是(　　)。
   A. tables_priv　　B. columns_priv　　C. db　　D. user
4. 以下账户命名错误的是(　　)。
   A. ''@''　　B. 'ab c'@'%'
   C. mark-manager@%　　D. test@localhost
5. 下列关于用户与权限的说法错误的是(　　)。
   A. 具有空白用户名的账户是匿名用户
   B. 通配符"%"和"_"都可以使用在用户的主机名中
   C. REVOKE ALL 回收的权限不包括 GRANT OPTION
   D. 以上说法都不正确

四、实训题

1. 请为用户名"xiaoming"密码"123abc"的用户授予查看 shop 数据库的权限。
2. 请创建"manager"用户并授予创建用户和删除用户的管理权限。

第 8 章 视　　图

学习目标

- 了解视图的概念和作用
- 掌握视图的创建、查看、修改和删除操作
- 掌握视图的数据操作

在前面章节的学习中，操作的数据表都是一些真实存在的表，其实，数据库还有一种虚拟表，它的结构和真实表一样，都是二维表，但是不存放数据，数据从真实表中获取，这种表被称为视图。本章将针对数据库中视图的基本操作进行详细讲解。

8.1 初识视图

8.1.1 视图的概念和使用

视图是从一个或多个表中导出来的表，它是一种虚拟存在的表，表的结构和数据都依赖于基本表。通过视图不仅可以看到存放在基本表中的数据，还可以像操作基本表一样，对数据进行查询、添加、修改和删除。

为了使读者直观看到视图的使用效果，下面通过一个案例进行演示。

```
#① 选择第 4 章创建的 shop 数据库
mysql>USE shop;
Database changed
#② 查询数据
mysql>SELECT id, name, price, price * 0.8 p FROM sh_goods LIMIT 3;
+----+---------+-------+---------+
| id | name    | price | p       |
+----+---------+-------+---------+
|1   |2B 铅笔  |  0.50 |   0.400 |
|2   |钢笔     | 15.00 |  12.000 |
|3   |碳素笔   |  1.00 |   0.800 |
+----+---------+-------+---------+
3 rows in set (0.00 sec)
#③ 创建 view_goods 视图 (CREATE VIEW 视图名 AS SELECT 语句)
mysql>CREATE VIEW view_goods AS
    ->SELECT id, name, price, price * 0.8 p FROM sh_goods LIMIT 3;
Query OK, 0 rows affected (0.00 sec)
```

```
#④ 查询视图
mysql> SELECT * FROM view_goods;
+----+--------+-------+--------+
| id | name   | price | p      |
+----+--------+-------+--------+
|1   |2B 铅笔 |  0.50 |  0.400 |
|2   |钢笔    | 15.00 | 12.000 |
|3   |碳素笔  |  1.00 |  0.800 |
+----+--------+-------+--------+
3 rows in set (0.00 sec)
#⑤ 删除视图(DROP VIEW 视图名)
mysql> DROP VIEW view_goods;
Query OK, 0 rows affected (0.00 sec)
```

在上述示例中，第③步用于创建 view_goods 视图，语法格式为“CREATE VIEW 视图名 AS SELECT 语句”，此处将 AS 后面的 SELECT 语句指定为第②步的查询语句。

将视图创建出来后，第④步使用 SELECT 语句查询 view_goods 视图，会看到其查询结果与第②步相同。值得一提的是，第③步的 SELECT 语句中的 WHERE、LIMIT 等子句也可以移动到第④步中使用，不影响查询结果，示例 SQL 语句如下。

```
#在创建视图的 SELECT 语句中移除 LIMIT 子句
mysql> CREATE VIEW view_goods AS
    -> SELECT id, name, price, price * 0.8 FROM sh_goods;
#在查询视图时使用 LIMIT 子句
mysql> SELECT * FROM view_goods LIMIT 3;
```

在查询视图时，SELECT 字段列表和 WHERE 等子句中的字段，只能使用创建视图时指定的 SELECT 语句中的字段，即 id、name、price 和 p 字段，而 sh_goods 表中的其他字段则无法通过 view_goods 视图查询。示例 SQL 语句如下。

```
#错误情景 1:字段列表中不能使用 stock 字段
mysql> SELECT stock FROM view_goods;
ERROR 1054 (42S22): Unknown column 'stock' in 'field list'
#错误情景 2:WHERE 子句中不能使用 stock 字段
mysql> SELECT * FROM view_goods WHERE stock = 0;
ERROR 1054 (42S22): Unknown column 'stock' in 'where clause'
```

通过上述案例可知，与直接操作基本表相比，视图具有如下优点。

(1) 简化查询语句。通过视图可以简化查询语句，简化用户的查询操作，使查询更加快捷。日常开发中可以将经常使用的查询定义为视图，从而避免大量重复的操作。

(2) 安全性。通过视图可以更方便地进行权限控制，能够使特定用户只能查询和修改他们所能见到的数据，数据库中的其他数据则既看不到也取不到。

(3) 逻辑数据独立性。视图可以屏蔽真实表结构变化带来的影响。例如，当其他应用程序查询数据时，若直接查询数据表，一旦表结构发生改变，查询的 SQL 语句就会发生改变，应用程序也必须随之更改。但若为应用程序提供视图，修改表结构后只需修改视图对应的 SELECT 语句，就无须更改应用程序。

8.1.2 创建视图的语法格式

创建视图使用 CREATE VIEW 语句，该语句的基本语法格式如下。

```
CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

从上述语法格式可以看出，创建视图的语句是由多条子句构成的。下面对语法格式中的每个部分进行解释，具体如下。

(1) CREATE：表示创建视图的关键字。

(2) OR REPLACE：可选，表示替换已有视图。

(3) ALGORITHM：可选，表示视图算法，会影响查询语句的解析方式，它的取值有如下 3 个，一般情况下使用 UNDEFINED 即可。

- UNDEFINED(默认)：由 MySQL 自动选择算法。
- MERGE：将 select_statement 和查询视图时的 SELECT 语句合并起来查询。
- TEMPTABLE：先将 select_statement 的查询结果存入临时表，然后用临时表进行查询。

(4) DEFINER：可选，表示定义视图的用户，与安全控制有关，默认为当前用户。

(5) SQL SECURITY：可选，用于视图的安全控制，它的取值有如下两个。

- DEFINER(默认)：由定义者指定的用户的权限来执行。
- INVOKER：由调用视图的用户的权限来执行。

(6) view_name：表示要创建的视图名称。

(7) column_list：可选，用于指定视图中的各个列的名称。默认情况下，与 SELECT 语句查询的列相同。

(8) AS：表示视图要执行的操作。

(9) select_statement：一个完整的查询语句，表示从某些表或视图中查出某些满足条件的记录，将这些记录导入视图中。

(10) WITH CHECK OPTION：可选，用于视图数据操作时的检查条件。若省略此子句，则不进行检查。它的取值有如下两个。

- CASCADED(默认)：操作数据时要满足所有相关视图和表定义的条件。例如，当在一个视图的基础上创建另一个视图时，进行级联检查。
- LOCAL：操作数据时满足该视图本身定义的条件即可。

小提示：

(1) 在默认情况下，新创建的视图保存在当前选择的数据库中。若要明确指定在某个数据库中创建视图，在创建时应将名称指定为“数据库名.视图名”。

(2) 在 SHOW TBLES 的查询结果中会包含已经创建的视图。

(3) 创建视图要求用户具有 CREATE VIEW 权限，以及查询涉及的列的 SELECT 权

限。如果还有OR REPLACE子句，必须具有视图的DROP权限。

(4) 在同一个数据库中，视图名称和已经存在的表名称不能相同，为了区分，建议在命名时添加“view_”前缀或“_view”后缀。

(5) 视图创建后，MySQL就会在数据库目录中创建一个“视图名.frm”文件。

8.2 视图管理

在上一节介绍了视图的概念和基本语法格式。在对视图有了初步的了解以后，本节将针对视图的创建、查看、修改以及删除操作进行详细讲解。

8.2.1 创建视图

在创建视图时，除了按照8.1.1节演示的方式进行创建，还可以创建多表视图、自定义视图中的列名称、进行安全控制等，下面将进行详细讲解。

1. 在多表上创建视图

除了在单表上创建视图，还可以在两个或者两个以上的基本表上创建视图。下面通过案例演示在sh_goods和sh_goods_category两张表上创建视图。

```
#① 创建视图
mysql> CREATE VIEW view_goods_cate AS
    -> SELECT g.id, g.name, c.name category_name FROM sh_goods
    -> LEFT JOIN sh_goods_category c
    -> ON g.category_id = c.id;
Query OK, 0 rows affected (0.01 sec)
#② 查询视图
mysql> SELECT * FROM view_goods_cate LIMIT 3;
+----+--------+---------------+
| id | name   | category_name |
+----+--------+---------------+
| 1  | 2B铅笔 | 文具          |
| 2  | 钢笔   | 文具          |
| 3  | 碳素笔 | 文具          |
+----+--------+---------------+
3 rows in set (0.00 sec)
```

从上述案例可以看出，当在创建视图时指定的SELECT语句涉及多张表的查询时，创建的视图就是多表视图。

2. 自定义列名称

通过创建视图的语法格式可知，视图的列名称可以自定义。下面通过案例演示在创建视图时自定义列名称，具体SQL语句和运行结果如下。

```
#① 创建视图
mysql> CREATE VIEW view_goods_promo (sn, title, promotion_price) AS
```

```
    ->SELECT id, name, price * 0.8 FROM sh_goods;
Query OK, 0 rows affected (0.01 sec)
#② 查询视图
mysql>SELECT * FROM view_goods_promo WHERE sn <=3;
+----+--------+------------------+
| sn | title  | promotion_price  |
+----+--------+------------------+
|1   | 2B 铅笔 |       0.400      |
|2   | 钢笔    |      12.000      |
|3   | 碳素笔  |       0.800      |
+----+--------+------------------+
3 rows in set (0.00 sec)
```

从上述结果可以看出，在创建视图时，自定义列名称的顺序与 AS 后 SELECT 字段列表的顺序一致，即 sn 对应 id，title 对应 name，promotion_price 对应 price * 0.8。需要注意的是，自定义列名称的数量必须与 SELECT 字段列表的数量一致，若不一致，MySQL 会报错，视图将无法创建。

3. 视图安全控制

从创建视图的语法格式可知，在创建视图时指定 DEFINER 和 SQL SECURITY 可以控制视图的安全。为了使读者更好地理解，下面通过案例演示它们的作用，具体如下。

```
#① 创建测试用户 shop_test
mysql>CREATE USER shop_test;
Query OK, 0 rows affected (0.00 sec)
#② 创建第 1 个视图,权限控制使用默认值
mysql>CREATE VIEW view_goods_t1 AS
    ->SELECT id, name FROM sh_goods LIMIT 1;
Query OK, 0 rows affected (0.00 sec)
#③ 创建第 2 个视图,设置 DEFINER 为 shop_test 用户
mysql>CREATE DEFINER='shop_test' VIEW view_goods_t2 AS
    ->SELECT id, name FROM sh_goods LIMIT 1;
Query OK, 0 rows affected (0.01 sec)
#④ 创建第 3 个视图,设置 SQL SECURITY 为 INVOKER
mysql>CREATE SQL SECURITY INVOKER VIEW view_goods_t3 AS
    ->SELECT id, name FROM sh_goods LIMIT 1;
Query OK, 0 rows affected (0.01 sec)
#⑤ 为 shop_test 用户赋予前面创建的 3 个视图的 SELECT 权限
mysql>GRANT SELECT ON view_goods_t1 TO 'shop_test';
Query OK, 0 rows affected (0.00 sec)
mysql>GRANT SELECT ON view_goods_t2 TO 'shop_test';
Query OK, 0 rows affected (0.00 sec)
mysql>GRANT SELECT ON view_goods_t3 TO 'shop_test';
Query OK, 0 rows affected (0.00 sec)
```

在上述操作创建的 3 个视图中，view_goods_t1 的 DEFINER 为当前用户 root，SQL SECURITY 为 DEFINER；view_goods_t2 的 DEFINER 为 shop_test 用户，SQL SECURITY 同样也是 DEFINER；view_goods_t3 的 DEFINER 为当前用户 root，但 SQL

SECURITY 为 INVOKER。

接下来重新打开一个命令行窗口，通过 mysql -ushop_test 登录 shop_test 用户，然后执行如下操作来测试视图是否可用。

```
mysql>USE shop;
Database changed
#① 第1个视图的 DEFINER 为 root,该用户有 sh_goods 表的 SELECT 权限
mysql>SELECT * FROM view_goods_t1;
+----+--------+
| id  | name     |
+----+--------+
|1    |2B 铅笔   |
+----+--------+
1 row in set (0.00 sec)
#② 第2个视图的 DEFINER 为 shop_test,该用户没有 sh_goods 表的 SELECT 权限
mysql>SELECT * FROM view_goods_t2;
ERROR 1356 (HY000): View 'shop.view_goods_t2' references invalid
table(s) or coumn(s) or function(s) or definer/invoker of view lack
rights to use them
#③ 第3个视图的 SQL SECURITY 为 INVOKER,
#会判断当前用户 shop_test 有无 sh_goods 表的 SELECT 权限
mysql>SELECT * FROM view_goods_t3;
ERROR 1356 (HY000): View 'shop.view_goods_t3' references invalid
table(s) or column(s) or function(s) or definer/invoker of view lack
rights to use them
```

从测试结果可以看出，只有 view_goods_t1 查询成功，这是因为该视图使用的 root 用户有权限查询基本表 sh_goods，而 view_goods_t2 和 view_goods_t3 使用的 shop_test 用户没有对基本表 sh_goods 的查询权限，因此查询失败。

8.2.2 查看视图

查看视图，是指查看数据库中已经存在的视图的定义。查看视图必须要有 SHOW VIEW 的权限。查看视图的方式有 3 种，具体如下。

1. 查看视图的字段信息

MySQL 提供的 DESCRIBE(DESC)语句不仅可以查看数据表的字段信息，还可以查看视图的字段信息。具体使用示例如下。

```
mysql>DESC view_goods_cate;
+---------------+------------------+------+-----+---------+-------+
| Field         | Type             | Null | Key | Default | Extra |
+---------------+------------------+------+-----+---------+-------+
| id            | int(10) unsigned | NO   |     | 0       |       |
| name          | varchar(120)     | NO   |     |         |       |
| category_name | varchar(100)     | YES  |     |         |       |
+---------------+------------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

2. 查看视图状态信息

MySQL 提供的 SHOW TABLE STATUS 语句不仅可以查看数据表的状态信息，还可以查看视图的状态信息。具体使用示例如下。

```
mysql> SHOW TABLE STATUS LIKE 'view_goods_cate' \G
*************************** 1. row ***************************
              Name: view_goods_cate
          Engine: NULL
        Version: NULL
   ⋮ (此处省略一些显示结果)
  Create_options: NULL
        Comment: VIEW
1 row in set (0.00 sec)
```

上述执行结果显示了 view_goods_cate 视图的状态信息，除 Name 和 Comment 之外，其他信息为 NULL。Comment 的值为 VIEW，表示所查的 view_goods_cate 是一个视图。

3. 查看视图的创建语句

使用 SHOW CREATE VIEW(或 SHOW CREATE TABLE)语句可以查看创建视图时的定义语句以及视图的字符编码。具体使用示例如下。

```
mysql> SHOW CREATE VIEW view_goods_cate \G
*************************** 1. row ***************************
                View: view_goods_cate
        Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`local
host` SQL SECURITY DEFINER VIEW `view_goods_cate` AS select `g`.`id` AS
`id`,`g`.`name` AS `name`,`c`.`name` AS `category_name` from (`sh_goods`
`g` left join `sh_goods_category` `c` on((`g`.`category_id` = `c`.`id`)))
character_set_client: gbk
collation_connection: gbk_chinese_ci
1 row in set (0.00 sec)
```

从上述执行结果可以看出，使用 SHOW CREATE VIEW 语句查看了视图的名称、创建语句、字符编码等信息。

8.2.3 修改视图

修改视图是指修改数据库中存在的视图的定义。例如，当基本表中的某些字段发生变化时，视图必须修改才能正常使用。在 MySQL 中修改视图的方式有两种，具体如下。

1. 替换已有的视图

通过 CREATE OR REPLACE VIEW 语句可以在创建视图时替换已有的同名视图，如果视图不存在，则创建一个视图。具体使用示例如下。

```
#① 创建视图
mysql> CREATE VIEW view_goods AS
```

```
    ->SELECT id, name, price FROM sh_goods;
Query OK, 0 rows affected (0.01 sec)
#② 修改已有视图
mysql>CREATE OR REPLACE VIEW view_goods AS
    ->SELECT id, name FROM sh_goods;
Query OK, 0 rows affected (0.01 sec)
#③ 查看修改结果
mysql>DESC view_goods;
+--------+------------------+------+-----+---------+-------+
| Field  | Type             | Null | Key | Default | Extra |
+--------+------------------+------+-----+---------+-------+
| id     | int(10) unsigned | NO   |     | 0       |       |
| name   | varchar(120)     | NO   |     |         |       |
+--------+------------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

在上述操作中,第①步创建的 view_goods 视图包含 price 字段,而第②步中没有 price 字段,通过第③步的查询结果可知,view_goods 视图已被第②步操作修改成功。

2. 使用 ALTER VIEW 语句修改视图

使用 ALTER VIEW 语句可以修改视图,其基本语法格式如下。

```
ALTER [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name [(column_list)]
AS SELECT_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

上述语法格式中,ALTER 后面的各部分子句与 CREATE VIEW 语句中的子句含义相同。接下来演示 ALTER VIEW 语句的使用,具体示例如下。

```
#① 修改视图
mysql>ALTER VIEW view_goods AS SELECT id FROM sh_goods;
Query OK, 0 rows affected (0.01 sec)
#② 查看修改结果
mysql>DESC view_goods;
+--------+------------------+------+-----+---------+-------+
| Field  | Type             | Null | Key | Default | Extra |
+--------+------------------+------+-----+---------+-------+
| id     | int(10) unsigned | NO   |     | 0       |       |
+--------+------------------+------+-----+---------+-------+
1 row in set (0.00 sec)
```

从上述结果可以看出,view_goods 视图修改成功。

8.2.4　删除视图

当视图不再需要时,可以将其删除,在删除时不会删除基本表中的数据。删除一个或多

个视图使用 DROP VIEW 语句，该语句的基本语法格式如下。

```
DROP VIEW [IF EXISTS] view_name [, view_name1] …
```

在上述语法格式中，view_name 是要删除的视图的名称，视图名称可以添加多个，使用逗号隔开。下面通过案例演示 DROP VIEW 的使用。

```
#① 删除视图
mysql> DROP VIEW view_goods;
Query OK, 0 rows affected (0.00 sec)
#② 检查视图是否已被删除
mysql> SELECT * FROM view_goods;
ERROR 1146 (42S02): Table 'shop.view_goods' doesn't exist
```

从上述查询结果可以看出，删除 view_goods 视图后，查询结果显示该视图不存在，说明视图被删除成功。

8.3 视图数据操作

视图数据操作就是通过视图来查询、添加、修改或删除基本表中的数据。因为视图是一个虚拟表，不保存数据，当通过视图操作数据时，实际操作的是基本表中的数据。视图的查询数据操作在前面已经讲过，本节将对视图的添加、修改和删除数据操作进行详细讲解。

8.3.1 添加数据

使用 INSERT 语句可以通过视图向基本表添加数据，具体示例如下。

```
#① 创建视图
mysql> CREATE VIEW view_category AS
    -> SELECT id, name FROM sh_goods_category;
Query OK, 0 rows affected (0.01 sec)
#② 添加数据
mysql> INSERT INTO view_category VALUES (17, '图书');
Query OK, 1 row affected (0.00 sec)
#③ 查询添加后的数据
mysql> SELECT id, name FROM sh_goods_category WHERE id = 17;
+----+------+
| id | name |
+----+------+
| 17 | 图书 |
+----+------+
1 row in set (0.00 sec)
```

从上述操作可以看出，通过视图添加的数据实际保存在基本表中。

脚下留心

在进行视图数据操作时，如果遇到如下情况，操作可能会失败。

(1) 操作的视图定义在多个表上。

（2）没有满足视图的基本表对字段的约束条件。

（3）在定义视图的 SELECT 语句后的字段列表中使用了数学表达式或聚合函数。

（4）在定义视图的 SELECT 语句中使用了 DISTINCT、UNION、TOP、GROUP BY 或 HAVING 子句。

8.3.2 修改数据

使用 UPDATE 语句可以通过视图修改基本表中的数据，具体示例如下。

```
#① 修改数据
mysql> UPDATE view_category SET name = '家电' WHERE id = 17;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
#② 查询修改后的数据
mysql> SELECT id, name FROM sh_goods_category WHERE id = 17;
+----+------+
| id   | name    |
+----+------+
| 17   | 家电    |
+----+------+
1 row in set (0.00 sec)
```

从上述操作可以看出，通过视图可以修改基本表中的数据。

8.3.3 删除数据

使用 DELETE 语句可以通过视图删除基本表中的数据，具体示例如下。

```
#① 删除数据
mysql> DELETE FROM view_category WHERE id = 17;
Query OK, 1 row affected (0.00 sec)
#② 查询数据是否已经删除
mysql> SELECT id, name FROM sh_goods_category WHERE id = 17;
Empty set (0.00 sec)
```

从上述操作可以看出，通过视图可以删除基本表中的数据。

8.3.4 视图检查条件

在创建视图的语法格式中，WITH CHECK OPTION 子句用于在视图数据操作时进行条件检查。为了使读者更好地理解这个子句的作用，下面通过案例进行演示。

```
#① 创建第 1 个视图
mysql> CREATE VIEW view_cate_t1 AS
    -> SELECT id, name FROM sh_goods_category WHERE id < 30;
Query OK, 0 rows affected (0.00 sec)
#② 创建第 2 个视图，使用 CASCADED(级联)检查
mysql> CREATE VIEW view_cate_t2 AS
    -> SELECT id, name FROM view_cate_t1 WHERE id > 20
```

```
    ->WITH CHECK OPTION; #相当于:WITH CASCADED CHECK OPTION
Query OK, 0 rows affected (0.00 sec)
#③ 插入数据测试,id必须大于 20 小于 30 才可以插入成功
mysql>INSERT INTO view_cate_t2 VALUES (17, 'test');
ERROR 1369 (HY000): CHECK OPTION failed 'shop.view_cate_t2'
mysql>INSERT INTO view_cate_t2 VALUES (21, 'test');
Query OK, 1 row affected (0.00 sec)
mysql>INSERT INTO view_cate_t2 VALUES (30, 'test');
ERROR 1369 (HY000): CHECK OPTION failed 'shop.view_cate_t2'
#④ 创建第 3 个视图,使用 LOCAL(本视图,非级联)检查
mysql>CREATE VIEW view_cate_t3 AS
    ->SELECT id, name FROM view_cate_t1 WHERE id>20
    ->WITH LOCAL CHECK OPTION;
Query OK, 0 rows affected (0.00 sec)
#⑤ 插入数据测试,只需满足 id大于 20 即可插入
mysql>INSERT INTO view_cate_t3 VALUES (30, 'test');
Query OK, 1 row affected (0.00 sec)
```

通过上述操作可知,当创建视图时添加 WITH CHECK OPTION 后,在对视图进行更新时会进行条件检查。检查方式有两种:默认情况下使用 CASCADED,表示级联检查;若设为 LOCAL,则只检查本视图定义的条件。

8.4 动手实践:视图的应用

数据库的学习在于多看、多学、多思考、多动手,只有将理论与实际相结合,才能够体会数据库开发与管理的重要性,展现知识学习的价值与力量。接下来请结合本章所学的知识完成视图的应用。

【实践目标】

通过前面的学习,读者已经掌握了如何进行视图的基本操作。本节将通过一个应用案例让读者熟练掌握在实际开发中创建并使用视图的完整过程。

【实践需求】

在电子商务网站的后台录入数据时,为了添加或修改商品属性信息,需要查询出商品所属分类下的属性信息,以及这些属性对应的属性值信息。接下来在 shop 数据库中创建两个视图,具体要求如下。

- sh_view_attr:用于根据商品分类 id 查找所有属性信息。
- sh_view_goods_attr:用于根据商品 id 查找所有属性信息。

【手动实践】

1. 创建 sh_view_attr 视图

使用 CREATE VIEW 语句创建 sh_view_attr 视图,在创建前可以先编写并测试符合需求的 SELECT 语句。根据需求,创建 sh_view_attr 视图的 SQL 语句如下。

```
mysql>CREATE VIEW sh_view_attr AS
    ->SELECT a.sort sort1, a.name attr1, b.sort sort2, b.name attr2,
    ->b.category_id FROM sh_goods_attr a
    ->JOIN sh_goods_attr b ON a.id =b.parent_id
    ->WHERE b.parent_id !=0 ORDER BY a.sort ASC, b.sort ASC;
Query OK, 0 rows affected (0.01 sec)
```

使用 SELECT 语句查询 sh_view_attr 视图,具体 SQL 语句和执行结果如下。

```
mysql>SELECT * FROM sh_view_attr WHERE category_id =6;
+--------+-----------+-------+------------+-------------+
| sort1  | attr1     | sort2 | attr2      | category_id |
+--------+-----------+-------+------------+-------------+
|   0    | 基本信息  |   0   | 机身颜色   |      6      |
|   0    | 基本信息  |   1   | 输入方式   |      6      |
|   0    | 基本信息  |   2   | 操作系统   |      6      |
|   1    | 屏幕      |   0   | 屏幕尺寸   |      6      |
|   1    | 屏幕      |   1   | 屏幕材质   |      6      |
|   1    | 屏幕      |   2   | 分辨率     |      6      |
|   2    | 摄像头    |   0   | 前置摄像头 |      6      |
|   2    | 摄像头    |   1   | 后置摄像头 |      6      |
|   3    | 电池信息  |   0   | 电池容量   |      6      |
|   3    | 电池信息  |   1   | 是否可拆卸 |      6      |
+--------+-----------+-------+------------+-------------+
10 rows in set (0.00 sec)
```

从上述结果可以看出,使用 sh_view_attr 视图成功查询出了分类 id 为 6 的所有属性信息,并按照 sort 字段进行了排序。

2. 创建 sh_view_goods_attr 视图

使用 CREATE VIEW 语句创建 sh_view_goods_attr 视图,具体 SQL 语句如下。

```
mysql>CREATE VIEW sh_view_goods_attr AS
    ->SELECT b.name attr1, a.name attr2, c.attr_value, c.goods_id
    ->FROM sh_goods_attr a
    ->JOIN sh_goods_attr b ON a.parent_id =b.id
    ->JOIN sh_goods_attr_value c ON a.id =c.attr_id
    ->WHERE a.parent_id !=0 ORDER BY b.sort ASC, a.sort ASC;
Query OK, 0 rows affected (0.01 sec)
```

使用 SELECT 语句查询 sh_view_goods_attr 视图,具体 SQL 语句和执行结果如下。

```
mysql>SELECT * FROM sh_view_goods_attr WHERE goods_id =5;
+----------+------------+------------+----------+
| attr1    | attr2      | attr_value | goods_id |
+----------+------------+------------+----------+
| 基本信息 | 机身颜色   | 黑色       |    5     |
| 基本信息 | 输入方式   | 触摸屏     |    5     |
| 基本信息 | 操作系统   | Android    |    5     |
```

```
| 屏幕       | 屏幕尺寸     | 5.5寸        |     5     |
| 屏幕       | 屏幕材质     | IPS          |     5     |
| 屏幕       | 分辨率       | 1920*1080    |     5     |
| 摄像头     | 前置摄像头   | 1600万       |     5     |
| 摄像头     | 后置摄像头   | 800万        |     5     |
| 电池信息   | 电池容量     | 3500mAh      |     5     |
| 电池信息   | 是否可拆卸   | 否           |     5     |
+-----------+-------------+-------------+----------+
10 rows in set (0.00 sec)
```

从上述结果可以看出,使用 sh_view_goods_attr 视图成功查询出了商品 id 为 5 的所有属性信息。

8.5 本章小结

本章主要讲解了创建视图、查看视图、修改视图、更新视图、删除视图,以及最后的视图应用案例。通过本章的学习,读者应该掌握如何创建视图,当基本表的字段发生变化时如何修改视图,以及如何通过视图修改基本表中的数据信息等知识。

8.6 课后练习

一、填空题

1. 视图是从一个或多个表中导出来的表,它的数据依赖于________。
2. 在 MySQL 中,创建视图使用________语句。
3. 在 MySQL 中,删除视图使用________语句。
4. 使用________语句可以查看创建视图时的定义语句。
5. 视图在数据库的三级模式中对应的是________模式。

二、判断题

1. 查看视图必须要有 SHOW VIEW 权限。(　　)
2. CREATE OR REPLACE VIEW 语句不会替换已经存在的视图。(　　)
3. 视图中不能包含基本表中被定义为非空的列。(　　)
4. 删除视图时,也会删除所对应基本表中的数据。(　　)
5. DROP 语句一次只能删除一个视图。(　　)

三、选择题

1. 创建视图应当具备的权限包括(　　)。
   A. CREATE VIEW　　B. USE VIEW
   C. SHOW VIEW　　D. CREATE TABLE
2. 下列选项中,用于查看视图的字段信息的语句是(　　)。
   A. DESCRIBE　　B. CREATE　　C. SHOW　　D. SELECT

3. 下列选项中，对视图中数据的操作包括(　　)。

A. 定义视图　　B. 修改数据　　C. 查看数据　　D. 删除数据

4. 下列关于视图优点的描述中，正确的是(　　)。

A. 实现了逻辑数据独立性

B. 提高安全性

C. 简化查询语句

D. 屏蔽真实表结构变化带来的影响

5. 下列关于视图创建的说法中，正确的是(　　)。

A. 可以建立在单表上

B. 可以建立在两张表基础上

C. 可以建立在两张或两张以上的表基础上

D. 以上都有可能

四、简答题

1. 请简述视图和基本表的区别。

2. 请简述修改视图的两种方式，并写出其基本语法。

五、实训题

1. 在 shop 数据库中创建 view_goods 视图，以 spu 为单位统计商品库存量。

2. 在 mydb 数据库中创建 student 数据表，表中有 id、name(学生姓名)、math(数学成绩)、chinese(语文成绩)和 english(英语成绩) 字段。然后创建视图 view_score，视图中包含 math、chinese、english 和 total(总分数)字段。

第 9 章 事 务

学习目标

- 理解事务的概念和 4 个基本特性
- 掌握事务的开启、提交和回滚操作
- 掌握事务的 4 种隔离级别

在实际开发中，对于复杂的数据操作过程，往往需要通过一组 SQL 语句来完成，这就必须保证所有命令执行的同步性。针对这样的情况，就需要使用 MySQL 中提供的事务处理机制。本章将针对事务的概念和使用等知识进行详细讲解。

9.1 事务处理

事务处理在数据库开发过程中有着非常重要的作用，它可以保证在同一个事务中的操作具有同步性。本节将针对事务处理的基础知识进行讲解。

9.1.1 事务的概念

现实生活中，人们经常会进行转账操作，转账可以分为转入和转出两部分，只有这两个部分都完成才认为转账成功。在数据库中，这个过程是使用两条 SQL 语句来实现的，如果其中任意一条语句出现异常没有执行，则会导致两个账户的金额不同步，造成错误。为了防止上述情况的发生，就需要使用 MySQL 中的事务（Transaction）。

在 MySQL 中，事务就是针对数据库的一组操作，它可以由一条或多条 SQL 语句组成，且每个 SQL 语句是相互依赖的。只要在程序执行过程中有一条 SQL 语句执行失败或发生错误，则其他语句都不会执行。也就是说，事务的执行要么成功，要么就返回到事务开始前的状态，这就保证了同一事务操作的同步性和数据的完整性。

MySQL 中的事务必须满足 A、C、I、D 这 4 个基本特性，具体如下。

（1）原子性（Atomicity）。原子性是指一个事务必须被视为一个不可分割的最小工作单元，只有事务中所有的数据库操作都执行成功，才算整个事务执行成功。事务中如果有任何一个 SQL 语句执行失败，已经执行成功的 SQL 语句也必须撤销，数据库的状态退回到执行事务前的状态。

（2）一致性（Consistency）。一致性是指在事务处理时，无论执行成功还是失败，都要保证数据库系统处于一致的状态，保证数据库系统不会返回到一个未处理的事务中。MySQL

中的一致性主要由日志机制实现，通过日志记录数据库的所有变化，为事务恢复提供了跟踪记录。

(3) 隔离性(Isolation)。隔离性是指当一个事务在执行时，不会受到其他事务的影响。保证了未完成事务的所有操作与数据库系统的隔离，直到事务完成为止，才能看到事务的执行结果。隔离性相关的技术有并发控制、可串行化、锁等。当多个用户并发访问数据库时，数据库为每一个用户开启的事务，不能被其他事务的操作数据所干扰，多个并发事务之间要相互隔离。

(4) 持久性(Durability)。持久性是指事务一旦提交，其对数据库的修改就是永久性的。需要注意的是，事务的持久性不能做到百分百的持久，只能从事务本身的角度来保证永久性，而一些外部原因导致数据库发生故障，如硬盘损坏，那么所有提交的数据可能都会丢失。

9.1.2 事务的基本操作

在默认情况下，用户执行的每一条 SQL 语句都会被当成单独的事务自动提交。如果要将一组 SQL 语句作为一个事务，则需要先执行以下语句显式地开启一个事务。

```
START TRANSACTION;
```

上述语句执行后，每一条 SQL 语句不再自动提交，用户需要使用以下语句手动提交，只有事务提交后，其中的操作才会生效。

```
COMMIT;
```

如果不想提交当前事务，可以使用如下语句取消事务(即回滚)。

```
ROLLBACK;
```

需要注意的是，ROLLBACK 只能针对未提交的事务回滚，已提交的事务无法回滚。当执行 COMMIT 或 ROLLBACK 后，当前事务就会自动结束。

为了让读者更好地学习事务，接下来通过一个转账的案例来演示如何使用事务。选择第 4 章创建的 shop 数据库，查看 sh_user 表中的用户数据，具体操作如下。

```
#① 选择数据库
mysql>USE shop;
#② 查看用户数据
mysql>SELECT name, money FROM sh_user;
+------+---------+
| name | money   |
+------+---------+
| Alex | 1000.00 |
| Bill | 1000.00 |
+------+---------+
2 rows in set (0.00 sec)
```

接下来开启一个事务，通过 UPDATE 语句将 Alex 用户的 100 元钱转给 Bill 用户，最

后提交事务，具体操作如下。

```
#① 开启事务
mysql> START TRANSACTION;
#② Alex 减少 100 元
mysql> UPDATE sh_user SET money = money - 100 WHERE name = 'Alex';
#③ Bill 增加 100 元
mysql> UPDATE sh_user SET money = money + 100 WHERE name = 'Bill';
#④ 提交事务
mysql> COMMIT;
```

上述操作完成后，使用 SELECT 语句查询 Alex 和 Bill 的金额，结果如下。

```
mysql> SELECT name, money FROM sh_user;
+-------+---------+
| name  | money   |
+-------+---------+
| Alex  |  900.00 |
| Bill  | 1100.00 |
+-------+---------+
2 rows in set (0.00 sec)
```

从查询结果可以看出，通过事务成功地完成了转账功能。接下来测试事务的回滚，开启事务后，将 Bill 的金额扣除 100 元，具体操作如下。

```
#① 开启事务
mysql> START TRANSACTION;
#② Bill 扣除 100 元
mysql> UPDATE sh_user SET money = money - 100 WHERE name = 'Bill';
#③ 查询 Bill 的金额
mysql> SELECT name, money FROM sh_user WHERE name = 'Bill';
+-------+---------+
| name  | money   |
+-------+---------+
| Bill  | 1000.00 |
+-------+---------+
1 row in set (0.00 sec)
```

上述操作完成后，执行回滚操作，然后查询 Bill 的金额，结果如下。

```
#① 回滚事务
mysql> ROLLBACK;
#② 查看 Bill 的金额
mysql> SELECT name, money FROM sh_user WHERE name = 'Bill';
+-------+---------+
| name  | money   |
+-------+---------+
| Bill  | 1100.00 |
+-------+---------+
1 row in set (0.00 sec)
```

从查询结果可以看出，Bill 的金额又恢复成 1100 元，说明事务回滚成功。

脚下留心

（1） MySQL 中的事务不允许嵌套，若在执行 START TRANSACTION 语句前上一个事务还未提交，会隐式地执行提交操作。

（2） 事务处理主要是针对数据表中数据的处理，不包括创建或删除数据库、数据表，修改表结构等操作，而且执行这类操作时会隐式地提交事务。

（3） MySQL 5.7 默认的存储引擎为 InnoDB，该存储引擎支持事务，而另一个常见的存储引擎 MyISAM 不支持事务。对于 MyISAM 存储引擎的数据表，无论事务是否提交，对数据的操作都会立即生效，不能回滚。

（4） 在 MySQL 中，还可以使用 START TRANSACTION 的别名 BEGIN 或 BEGIN WORK 来显式地开启一个事务。但由于 BEGIN 与 MySQL 编程中的 BEGIN…END 冲突，因此不推荐使用 BEGIN。

多学一招：事务的自动提交

MySQL 默认是自动提交模式，如果没有显式开启事务(START TRANSACTION)，每一条 SQL 语句都会自动提交(COMMIT)。如果用户想要控制事务的自动提交方式，可以通过更改 AUTOCOMMIT 变量来实现，将其值设为 1 表示开启自动提交，设为 0 表示关闭自动提交。若要查看当前会话的 AUTOCOMMIT 值，使用如下语句。

```
mysql> SELECT @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            1 |
+--------------+
1 row in set (0.00 sec)
```

从查询结果可以看出，当前会话开启了事务的自动提交。若要关闭当前会话的事务自动提交，可以使用如下语句。

```
SET AUTOCOMMIT = 0;
```

上述语句执行后，用户需要手动执行提交(COMMIT)操作，才会提交事务。否则，若直接终止 MySQL 会话，MySQL 会自动进行回滚。

9.1.3 事务的保存点

在回滚事务时，事务内所有的操作都将撤销。而若希望只撤销一部分，可以用保存点来实现。使用以下语句可以在事务中设置一个保存点。

```
SAVEPOINT 保存点名;
```

在设置保存点后，使用以下语句可以将事务回滚到指定保存点。

```
ROLLBACK TO SAVEPOINT 保存点名;
```

使用时若不再需要一个保存点,可以使用如下语句删除保存点。

```
RELEASE SAVEPOINT 保存点名;
```

值得一提的是,一个事务中可以创建多个保存点,在提交事务后,事务中的保存点就会被删除。另外,在回滚到某个保存点后,在该保存点之后创建过的保存点也会消失。

接下来通过案例演示事务保存点的使用。首先查询 Alex 的金额,如下所示。

```
mysql> SELECT name, money FROM sh_user WHERE name = 'Alex';
+-------+--------+
| name  | money  |
+-------+--------+
| Alex  | 900.00 |
+-------+--------+
1 row in set (0.00 sec)
```

从查询结果可知,Alex 当前金额为 900 元。然后开启事务,将 Alex 的金额扣除 100 元后,创建保存点 s1,再将 Alex 的金额扣除 50 元,如下所示。

```
#① 开启事务
mysql> START TRANSACTION;
#② Alex 扣除 100 元
mysql> UPDATE sh_user SET money = money - 100 WHERE name = 'Alex';
#③ 创建保存点 s1
mysql> SAVEPOINT s1;
#④ Alex 再扣除 50 元
mysql> UPDATE sh_user SET money = money - 50 WHERE name = 'Alex';
```

完成上述操作后,将事务回滚到保存点 s1,然后查询 Alex 的金额,如下所示。

```
#① 回滚到保存点 s1
mysql> ROLLBACK TO SAVEPOINT s1;
#② 查询 Alex 的金额
mysql> SELECT name, money FROM sh_user WHERE name = 'Alex';
+-------+--------+
| name  | money  |
+-------+--------+
| Alex  | 800.00 |
+-------+--------+
1 row in set (0.00 sec)
```

在上述结果中,Alex 的金额只减少了 100 元,说明当前恢复到了保存点 s1 时的数据状态。再次执行回滚操作,则恢复到事务开始时的状态,如下所示。

```
#① 回滚事务
mysql> ROLLBACK;
#② 查看 Alex 的金额
mysql> SELECT name, money FROM sh_user WHERE name = 'a';
```

```
+-------+--------+
| name  | money  |
+-------+--------+
| Alex  | 900.00 |
+-------+--------+
1 row in set (0.00 sec)
```

上述结果显示的金额与事务开始时的金额相同,说明事务回滚成功。

多学一招:控制事务结束后的行为

事务的提交(COMMIT)和回滚(ROLLBACK)还有一些可选子句,如下所示。

```
COMMIT [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK  [AND [NO] CHAIN] [[NO] RELEASE]
```

在上述选项中,AND CHAIN 用于在当前事务结束时,立即启动一个新事务,并且新事务与刚结束的事务有相同的隔离级;RELEASE 用于在终止当前事务后,让服务器断开与客户端的连接。若添加 NO,则表示抑制 CHAIN 和 RELEASE 完成。

9.2　事务隔离级别

由于数据库是一个多用户的共享资源,MySQL 允许多线程并发访问,因此用户可以通过不同的线程执行不同的事务。为了保证这些事务之间不受影响,对事务设置隔离级是十分必要的。本节将针对事务的隔离级别进行详细讲解。

9.2.1　查看隔离级别

对于隔离级的查看,MySQL 提供了以下几种不同的方式,具体使用哪种方式查询还需根据实际需求进行选择,具体如下。

```
#① 查看全局隔离级
SELECT @@global.transaction_isolation;
#② 查看当前会话中的隔离级
SELECT @@session.transaction_isolation;
#③ 查看下一个事务的隔离级
SELECT @@transaction_isolation;
```

在以上语句中,全局的隔离级影响的是所有连接 MySQL 的用户,而当前会话的隔离级别只影响当前正在登录 MySQL 服务器的用户,不会影响其他用户。而下一个事务的隔离级别仅对当前用户的下一个事务操作有影响。

在默认情况下,上述 3 种方式返回的结果都是 REPEATABLE-READ,表示隔离级别为可重复读。以第 3 种方式为例,查询结果如下所示。

```
mysql> SELECT @@transaction_isolation;
+-------------------------+
| @@transaction_isolation |
```

```
+--------------------------+
| REPEATABLE-READ          |
+--------------------------+
1 row in set (0.00 sec)
```

除了以上的 REPEATABLE READ(可重复读)外，MySQL 中事务的隔离级别还有 READ UNCOMMITTED(读取未提交)、READ COMMITTED(读取提交)和 SERIALIZABLE(可串行化)。具体内容会在后面的小节中详细讲解。

9.2.2 修改隔离级别

在 MySQL 中，事务的隔离级别可以通过 SET 语句进行设置，具体语法如下。

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL 参数值
```

在上述语法中，SET 后的 SESSION 表示当前会话，GLOBAL 表示全局，若省略表示设置下一个事务的隔离级。TRANSACTION 表示事务，ISOLATION 表示隔离，LEVEL 表示级别。参数值可以是 READ UNCOMMITTED、READ COMMITTED、REPEATABLE READ 或 SERIALIZABLE 中的一种。

下面演示将事务的隔离级别修改为 READ UNCOMMITTED，具体如下。

```
#① 修改事务隔离级别
mysql> SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
Query OK, 0 rows affected (0.00 sec)
#② 查看是否修改成功
mysql> SELECT @@session.transaction_isolation;
+--------------------------+
| @@session.transaction_isolation   |
+--------------------------+
| READ-UNCOMMITTED         |
+--------------------------+
1 row in set (0.00 sec)
```

从上述结果可以看出，当前事务的隔离级别已经修改为 READ UNCOMMITTED。

接下来将事务的隔离级别修改为默认的 REPEATABLE READ，具体如下。

```
#① 修改事务隔离级别
mysql> SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
Query OK, 0 rows affected (0.00 sec)
#② 查看是否修改成功
mysql> SELECT @@session.transaction_isolation;
+--------------------------+
| @@session.transaction_isolation   |
+--------------------------+
| REPEATABLE-READ          |
+--------------------------+
1 row in set (0.00 sec)
```

从上述结果可以看出，当前事务的隔离级别已经修改为 REPEATABLE READ。

多学一招：只读事务

默认情况下，事务的访问模式为 READ WRITE（读/写模式），表示事务可以执行读（查询）或写（更改、插入、删除等）操作。若开发需要，可以将事务的访问模式设置为 READ ONLY（只读模式），禁止对表进行更改。具体 SQL 语句如下。

```
#① 设置只读事务
SET [SESSION | GLOBAL] TRANSACTION READ ONLY
#② 恢复成读写事务
SET [SESSION | GLOBAL] TRANSACTION READ WRITE
```

9.2.3　MySQL 的 4 种隔离级别

MySQL 中事务隔离级别有 READ UNCOMMITTED（读取未提交）、READ COMMITTED（读取提交）、REPEATABLE READ（可重复读）和 SERIALIZABLE（可串行化）。下面针对每种隔离级别的特点、带来的问题以及解决方案进行详细讲解。

1. READ UNCOMMITTED（读取未提交）

READ UNCOMMITTED 是事务中最低的级别，在该级别下的事务可以读取到其他事务中未提交的数据，这种读取的方式也被称为脏读（Dirty Read）。简而言之，脏读是指一个事务读取了另外一个事务未提交的数据。

例如，Alex 要给 Bill 转账 100 元购买商品，Alex 开启事务后转账，但不提交事务，通知 Bill 来查询，如果 Bill 的隔离级别较低，就会读取到 Alex 的事务中未提交的数据，发现 Alex 确实给自己转了 100 元，就给 Alex 发货。等 Bill 发货成功后，Alex 将事务回滚，Bill 就会受到损失，这就是脏读造成的。

为了演示和解决上述情况，首先需要开启两个命令行窗口，分别登录到 MySQL 数据库，执行 USE shop 切换到 shop 数据库。然后使用这两个窗口分别模拟 Alex 和 Bill，以下称为客户端 A 和客户端 B。准备完成后，按照如下步骤进行操作。

（1）设置客户端 B 的事务隔离级别。由于 MySQL 默认的隔离级别 REPEATABLE READ 可以避免脏读，为了演示脏读，需要将客户端 B 的隔离级别设为较低的 READ UNCOMMITTED，具体如下。

```
#客户端 B
mysql> SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

（2）演示客户端 B 的脏读。在客户端 B 中查询 Bill 当前的金额，具体如下。

```
#客户端 B
mysql> SELECT name, money FROM sh_user WHERE name = 'Bill';
+-------+----------+
| name  | money    |
+-------+----------+
| Bill  | 1100.00  |
+-------+----------+
1 rows in set (0.00 sec)
```

在客户端 A 中开启事务,并执行转账操作,具体如下。

```
#客户端 A
mysql> START TRANSACTION;
mysql> UPDATE sh_user SET money = money - 100 WHERE name = 'Alex';
mysql> UPDATE sh_user SET money = money + 100 WHERE name = 'Bill';
```

此时客户端 A 未提交事务,客户端 B 查询金额,会看到金额已经增加,如下所示。

```
#客户端 B
mysql> SELECT name, money FROM sh_user WHERE name = 'Bill';
+-------+---------+
| name  | money   |
+-------+---------+
| Bill  | 1200.00 |
+-------+---------+
1 row in set (0.00 sec)
```

(3) 避免客户端 B 的脏读。将客户端 B 的事务隔离级设为 READ COMMITTED(或更高级别)可以避免脏读。设置后再次查询 Bill 的金额,如下所示。

```
#客户端 B
mysql> SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
mysql> SELECT name, money FROM sh_user WHERE name = 'Bill';
+-------+---------+
| name  | money   |
+-------+---------+
| Bill  | 1100.00 |
+-------+---------+
1 row in set (0.00 sec)
#客户端 A
mysql> ROLLBACK;
```

从上述结果可以看出,由于客户端 A 没有提交事务,客户端 B 读取到了客户端 A 提交前的结果,说明 READ COMMITTED 级别可以避免脏读。客户端 B 操作完成后,回滚客户端 A 的事务,以免影响后面的案例演示。

值得一提的是,脏读在实际应用中会带来很多问题,除非用户有很好的理由,否则,为了保证数据的一致性,在实际应用中几乎不会使用这个隔离级别。

2. READ COMMITTED(读取提交)

READ COMMITTED 是大多数 DBMS(如 SQL Server、Oracle)的默认隔离级,但不包括 MySQL。在该隔离级下只能读取其他事务已经提交的数据,避免了脏读数据的现象。但是在该隔离级别下,会出现不可重复读(NON-REPEATABLE READ)的问题。

不可重复读是指在一个事务中多次查询的结果不一致,原因是查询的过程中数据发生了改变。例如,在网站后台统计所有用户的总金额,第 1 次查询 Alex 有 900 元,为了验证查询结果,第 2 次查询 Alex 有 800 元,两次查询结果不同,原因是第 2 次查询前 Alex 取出了 100 元。

为了读者更好地理解，接下来就通过案例演示和解决以上不可重复读的情况。假设客户端 A 是 Alex 用户，客户端 B 是网站后台，具体操作步骤如下。

(1) 演示客户端 B 的不可重复读。当客户端 B 的事务隔离级为 READ COMMITTED 时，会出现不可重复读的情况。在客户端 B 中开启事务，查询 Alex 的金额，然后在客户端 A 中将 Alex 的金额扣除 100 元，最后客户端 B 再次查询 Alex 的金额，具体如下。

```
#客户端 B
mysql> SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
mysql> START TRANSACTION;
mysql> SELECT name, money FROM sh_user WHERE name = 'Alex';
+-------+---------+
| name  | money   |
+-------+---------+
| Alex  | 900.00  |
+-------+---------+
1 row in set (0.00 sec)
#客户端 A
mysql> UPDATE sh_user SET money = money - 100 WHERE name = 'Alex';
#客户端 B
mysql> SELECT name, money FROM sh_user WHERE name = 'Alex';
+-------+---------+
| name  | money   |
+-------+---------+
| Alex  | 800.00  |
+-------+---------+
1 row in set (0.00 sec)
mysql> COMMIT;
```

从上述结果可以看出，客户端 B 在同一个事务中两次查询的结果不一致，这就是不可重复读的情况。操作完成后，将客户端 B 的事务提交，以免影响后面的演示。

(2) 避免网站后台的不可重复读。将客户端 B 的事务隔离级别设为默认级别 REPEATABLE READ，可以避免不可重复读的情况。在该级别下按照上一步的方式重新测试，结果如下。

```
#客户端 B
mysql> SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
mysql> START TRANSACTION;
mysql> SELECT name, money FROM sh_user WHERE name = 'Alex';
+-------+---------+
| name  | money   |
+-------+---------+
| Alex  | 800.00  |
+-------+---------+
1 row in set (0.00 sec)
#客户端 A
mysql> UPDATE sh_user SET money = money + 100 WHERE name = 'Alex';
#客户端 B
mysql> SELECT name, money FROM sh_user WHERE name = 'Alex';
```

```
+-------+--------+
| name  | money  |
+-------+--------+
| Alex  | 800.00 |
+-------+--------+
1 row in set (0.00 sec)
mysql> COMMIT;
```

从上述结果可以看出，客户端 B 两次查询的结果是相同的，说明 REPEATABLE READ 可以避免不可重复读的情况。最后将客户端 B 的事务提交，以免影响后面的案例演示。

3. REPEATABLE READ(可重复读)

REPEATABLE READ 是 MySQL 的默认事务隔离级，它解决了脏读和不可重复读的问题，确保了同一事务的多个实例在并发读取数据时，会看到同样的结果。

但在理论上，该隔离级会出现幻读(PHANTOM READ)的现象。幻读又被称为虚读，是指在一个事务内两次查询中数据条数不一致，幻读和不可重复读有些类似，同样发生在两次查询过程中。不同的是，幻读是由于其他事务做了插入记录的操作，导致记录数有所增加。不过，MySQL 的 InnoDB 存储引擎通过多版本并发控制机制解决了幻读的问题。

例如，在网站后台统计所有用户的总金额时，当前只有两个用户，总金额为 2000 元，若在此时新增一个用户，并且存入 1000 元，再次统计时发现总金额变为 3000 元，造成了幻读的情况。

为了读者更好地理解，接下来就通过案例演示和解决以上幻读的情况。假设客户端 A 用于新增用户，客户端 B 用于统计金额，具体操作步骤如下。

(1) 演示客户端 B 的幻读。由于客户端 B 当前的隔离级别 REPEATABLE READ 可以避免幻读，因此需要将级别降低为 READ COMMITTED。降低后，开启事务，统计总金额，然后在客户端 A 中插入一条新记录 Tom 用户，再次统计总金额，具体如下。

```
#客户端 B
mysql> SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
mysql> START TRANSACTION;
mysql> SELECT SUM(money) FROM sh_user;
+------------+
| SUM(money) |
+------------+
|    2000.00 |
+------------+
1 row in set (0.00 sec)
#客户端 A
mysql> INSERT INTO sh_user (id, name, money) VALUES (3, 'Tom', 1000);
#客户端 B
mysql> SELECT SUM(money) FROM sh_user;
+------------+
| SUM(money) |
+------------+
|    3000.00 |
```

```
+------------+
1 row in set (0.00 sec)
mysql>COMMIT;
```

从上述结果可以看出，两次统计的结果不同，这就是幻读的情况。操作完成后，将客户端 B 的事务提交，以免影响后面的演示。

（2）避免客户端 B 的幻读。将客户端 B 的隔离级别设置为 REPEATABLE READ，即可避免幻读，结果如下。

```
#客户端 B
mysql>SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
mysql>START TRANSACTION;
mysql>SELECT SUM(money) FROM sh_user;
+------------+
| SUM(money)  |
+------------+
|   3000.00   |
+------------+
1 row in set (0.00 sec)
#客户端 A
mysql>INSERT INTO sh_user (id, name, money) VALUES (4, 'd', 1000);
#客户端 B
mysql>SELECT SUM(money) FROM sh_user;
+------------+
| SUM(money)  |
+------------+
|   3000.00   |
+------------+
1 row in set (0.00 sec)
mysql>COMMIT;
```

从上述结果可以看出，客户端 B 两次统计的结果是相同的，说明 REPEATABLE READ 级别可以避免幻读的情况。最后将客户端 B 的事务提交，以免影响后面的案例演示。

4. SERIALIZABLE（可串行化）

SERIALIZABLE 是最高级别的隔离级，它在每个读的数据行上加锁，使之不会发生冲突，从而解决了脏读、不可重复读和幻读的问题。但是由于加锁可能导致超时（Timeout）和锁竞争（Lock Contention）现象，因此 SERIALIZABLE 也是性能最低的一种隔离级。除非为了数据的稳定性，需要强制减少并发的情况时，才会选择此种隔离级。

为了读者更好地理解，接下来就通过案例演示超时的情况。假设客户端 B 执行查询操作，客户端 A 执行更新操作，具体操作步骤如下。

（1）演示可串行化。将客户端 B 的事务隔离级别设置为 SERIALIZABLE，然后开启事务，具体如下。

```
#客户端 B
mysql>SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
mysql>START TRANSACTION;
```

在客户端B开启事务后，查看Alex的金额，具体如下。

```
#客户端B
mysql> SELECT name, money FROM sh_user WHERE name = 'Alex';
+-------+--------+
| name  | money  |
+-------+--------+
| Alex  | 900.00 |
+-------+--------+
1 row in set (0.00 sec)
```

在客户端A中将Alex的金额增加100元，会发现UPDATE操作一直在等待，而不是立即处理成功，如下所示。

```
#客户端A
mysql> UPDATE sh_user SET money = money + 100 WHERE name = 'Alex';
(此时光标在不停闪烁,进入等待状态)
```

(2) 提交客户端B的事务。在客户端A的UPDATE操作等待时，提交客户端B的事务，客户端A的操作才会执行，提示执行结果，具体如下。

```
#客户端B
mysql> COMMIT;
#客户端A
Query OK, 1 row affected (3.99 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

若客户端B一直未提交事务，客户端A的操作会一直等待，直到超时后，出现如下提示信息，表示锁等待超时，尝试重新启动事务。

```
#客户端A
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

在默认情况下，锁等待的超时时间为50秒，可以通过如下语句查询。

```
mysql> SELECT @@innodb_lock_wait_timeout;
+----------------------------+
| @@innodb_lock_wait_timeout |
+----------------------------+
|                         50 |
+----------------------------+
1 row in set (0.00 sec)
```

从上述情况可以看出，如果一个事务使用了SERIALIZABLE隔离级别，在这个事务没有被提交前，其他会话只能等到当前操作完成后，才能进行操作，这样会非常耗时，而且会影响数据库的并发性能，所以通常情况下不会使用这种隔离级别。

9.3　动手实践：事务的应用

数据库的学习在于多看、多学、多思考、多动手，只有将理论与实际相结合，才能够体会数据库开发与管理的重要性，展现知识学习的价值与力量。接下来请结合本章所学的知识完成事务的应用。

【实践目标】

通过前面的学习，读者已经掌握了事务的概念和使用。本节将通过一个应用案例让读者熟练掌握在实际开发中利用事务解决问题。

【实践需求】

请利用事务实现在用户下订单后，订单商品表 sh_order_goods 中对应订单插入的商品数量大于实际商品库存量时，取消 sh_order_goods 表中数据的添加。

【动手实践】

（1）事务操作前，查看 sh_order_goods 订单商品表。

```
mysql> SELECT id, order_id, goods_id, goods_num FROM sh_order_goods;
Empty set (0.00 sec)
```

（2）开启事务。

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

（3）在事务中执行操作。

```
mysql> INSERT INTO sh_order_goods (order_id, goods_id, goods_num)
    -> VALUES (1, 4, 1);
Query OK, 1 row affected (0.00 sec)
mysql> UPDATE sh_goods SET stock = stock - 1 WHERE id = 4;
ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in
'(`shop`.`sh_goods`.`stock` - 1)'
```

以上的操作中，在向 sh_order_goods 订单商品表中插入数据后，需要同时修改 sh_goods 商品表中对应商品的库存量。例如，以上操作中订单内 id 为 4 的商品添加 1 个，但是在更新此商品的对应库存时报错，证明当前 sh_order_goods 表中对应商品的库存不符合实际情况。此时需要取消以上的所有操作，保证数据的一致性。

（4）回滚事务。

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
```

执行完以上的操作后，取消开启事务后执行的 INSERT 和 UPDATE 操作。

（5）事务操作后，查看 sh_order_goods 订单商品表。

```
mysql> SELECT id, order_id, goods_id, goods_num FROM sh_order_goods;
Empty set (0.00 sec)
```

9.4 本章小结

本章主要讲解了事务的概念、基本操作、保存点、隔离级别等内容，以及事务的应用案例。通过本章的学习，读者应掌握事务的应用场景，了解事务不同隔离级别的特点，具备运用事务解决实际需求的能力。

9.5 课后练习

一、填空题

1. 事务是针对________的一组操作。
2. 每个事务都是完整不可分割的最小单元是事务的________性。
3. 开启事务的语句是________。
4. 事务的自动提交通过________变量来控制。
5. 事务的 4 个隔离级别中性能最高的是________。

二、判断题

1. MySQL 中默认操作是自动提交模式。（　　）
2. 数据库的隔离级别越高，并发性能越低。（　　）
3. 只有多条 SQL 语句才能组成事务。（　　）
4. 已经提交的事务不能回滚。（　　）
5. 事务执行时间越短，并发性能越高。（　　）

三、选择题

1. MySQL 默认隔离级别为（　　）。
   A. READ UNCOMMITTED　　B. READ COMMITTED
   C. REPEATABLE READ　　D. SERIALIZABLE
2. 下列事务隔离级别中，可以避免脏读的有（　　）。
   A. READ UNCOMMITTED　　B. READ COMMITTED
   C. REPEATABLE READ　　D. SERIALIZABLE
3. 下列选项中会隐式提交事务的命令有（　　）。
   A. START TRANSACTION　　B. CREATE TABLE
   C. ALTER TABLE　　D. SELECT
4. 一个事务读取了另外一个事务未提交的数据，称为（　　）。
   A. 幻读　　B. 脏读　　C. 不可重复读　　D. 可串行化

5. 下列关于 MySQL 中事务的说法,错误的是(　　)。
   A. 事务就是针对数据库的一组操作
   B. 事务中的语句要么都执行,要么都不执行
   C. 事务提交后其中的操作才会生效
   D. 提交事务的语句为 SUBMIT

四、简答题

1. 请简述什么是事务。
2. 请简述什么是事务的 ACID 特性。

五、实训题

1. 请利用事务实现在用户下订单时,检查商品库存是否充足。
2. 请利用事务在用户下订单前,检测当前用户是否已被激活,若未激活,则需激活此用户后,才能再次下订单。

第 10 章 数据库编程

学习目标

- 熟悉内置函数以及自定义函数
- 掌握存储过程及变量的使用
- 熟悉流程控制及游标的用法
- 掌握特定事件的定时处理
- 了解预处理 SQL 语句的应用

MySQL 数据库除了可以增、删、改、查，还可以自定义函数、存储过程与变量，利用游标进行数据检索，触发事件，执行定时任务，预处理 SQL 语句等。例如，根据传递的参数进行判断，然后让符合条件的语句执行，利用变量保存处理后的数据等。本章将围绕数据库编程进行详细讲解。

10.1 函数

学习过编程语言的读者都了解，函数指的是一段用于完成特定功能的代码。当使用一个函数时，只需关心函数的参数和返回值，就可以完成一个特定的功能。为此，MySQL 中提供了大量的内置函数以及自定义函数功能供开发者使用，从而提高了用户对数据库和数据的管理和操作效率，灵活地满足不同用户的需求，这也是 MySQL 功能强大的一个重要因素。本节将针对 MySQL 中函数使用进行详细讲解。

10.1.1 内置函数

MySQL 提供的内置函数，也可以称为系统函数，这些函数无须定义，读者仅需根据实际需要传递参数直接调用即可。这些内置函数从功能方面划分，大致可以分为数学函数、数据类型转换函数、字符串函数、日期和时间函数、加密函数、系统信息函数、JSON 函数以及其他常用函数。

1. 数学函数

在本书第 5 章讲解算术运算符时，简单地列举了一小部分常用的数学函数，用于理解相关的算术运算。实际上，MySQL 提供的数学函数非常丰富，基本能够满足日常的数据库操作与管理。根据其使用的范围不同，大致可将其分为三角函数、指数函数、对数函数、求近似

值函数、进制函数等,具体如表 10-1 所示。

表 10-1　常用数学函数

分类	函数名称	描　　述
三角函数	PI()	计算圆周率
	RADIANS(x)	用于将角度 x 转换为弧度
	DEGREES(x)	用于将弧度 x 转换为角度
	SIN(x)	正弦函数
	COS(x)	余弦函数
	TAN(x)	正切函数
	COT(x)	余切函数
	ASIN(x)	反正弦函数
	ACOS(x)	反余弦函数
	ATAN(x)	反正切函数
指数函数	SQRT(x)	求 x 的平方根
	POW(x,y)或 POWER(x,y)	幂运算函数(计算 x 的 y 次方)
	EXP(x)	计算 e(自然对数的底约为 2.71828)的 x 次方
对数函数	LOG(x)	计算 x 的自然对数
	LOG10(x)	计算以 10 为底的对数
求近似值函数	ROUND(x,[y])	计算离 x 最近的整数;若设置参数 y,与 FORMAT(x,y)功能相同
	TRUNCATE(x,y)	返回小数点后保留 y 位的 x(舍弃多余小数位,不进行四舍五入)
	FORMAT(x,y)	返回小数点后保留 y 位的 x(进行四舍五入)
	CEIL(x)或 CEILING(x)	返回大于等于 x 的最小整数
	FLOOR(x)	返回小于等于 x 的最大整数
进制函数	BIN(x)	返回 x 的二进制数
	OCT(x)	返回 x 的八进制数
	HEX(x)	返回 x 的十六进制数
	ASCII(c)	返回字符 c 的 ASCII 码(ASCII 码介于 0～255)
	CHAR(c_1,c_2,c_3,…)	将 c_1,c_2,c_3,…的 ASCII 码转换为字符,然后返回这些字符组成的字符串
	CONV(x,$code_1$,$code_2$)	将 $code_1$ 进制的 x 变为 $code_2$ 进制数
其他函数	RAND()	默认返回[0,1]之间的随机数
	ABS(x)	获取 x 的绝对值
	MOD(x,y)	求模运算,与 x%y 的功能相同

表 10-1 根据不同的分类，归纳了 MySQL 中各类的数学函数及相关的释义。另外，由于数学函数的使用非常简单，其中需要注意的事项在第 5 章中已经讲解，这里不再赘述。

2. 数据类型转换函数

在数据库管理和操作时，经常需要对指定的数据类型转换后才能获取到想要的结果。例如，前面学习按中文数据排序时，就需要使用 CONVER(…USING…)将该字段的字符集指定为 gbk 后才能够正确的排序。常用的如表 10-2 所示。

表 10-2 数据类型转换函数

函数名称	描述
CONVERT(x,type)	以 type 类型返回 x
CONVERT(x USING 字符集)	以指定字符集返回 x 数据
CAST(x AS type)	以 type 类型返回 x
UNHEX(x)	将 x 转为十六进制数字，然后再转为由数字表示的字符

在表 10-2 中，CONVERT()和 CAST()函数的参数 x 可以是任何类型的表达式，参数 type 的可选值为 BINARAY、CHAR、DATE、DATETIME、DECIMAL、JSON、SIGNED [INTEGER]、TIME 和 UNSIGNED [INTEGER]。

为了读者更好地理解，下面通过案例演示数据类型转换函数的使用，具体 SQL 语句及执行结果如下。

```
mysql> SELECT CONVERT(3.6/-1.2,SIGNED) res1,
    -> CAST(0.5/1 AS UNSIGNED) res2,
    -> UNHEX('4D7953514C') res3;
+------+------+-------+
| res1 | res2 | res3  |
+------+------+-------+
|  -3  |   1  |MySQL  |
+------+------+-------+
1 row in set (0.00 sec)
```

上述操作中，CONVERT()将表达式 3.6/－1.2 的结果以有符号整型返回－3，CAST()将表达式 0.5/1 的结果以无符号整型返回 1，UNHEX()将字符串 4D7953514C 转为十六进制后再转为数字对应的字符。

另外，值得一提的是，在进行数据类型转换或字符集转换时，参数 x 原来的数据类型以及字符集并没有发生变化，而是生成一个新的指定类型或字符集的数据。

3. 字符串函数

MySQL 中对字符串的操作提供了很多的函数，可以获取字符串的长度、字节数、全部转成大写或小写、获取子串及位置、去除字符串两端的空格等。常用的如表 10-3 所示。

表 10-3　常用字符串函数

函数名称	描　　述
CHAR_LENGTH()	获取字符串的长度
LENGTH()	获取字符串占用的字节数
REPEAT()	重复指定次数的字符串，并保存到一个新字符串中
SPACE()	重复指定次数的空格，并保存到一个新字符串中
UPPER()	将字符串全部转为大写字母，与 UCASE()函数等价
LOWER()	将字符串全部转为小写字母，与 LCASE()函数等价
STRCMP()	比较两个字符串的大小
REVERSE()	颠倒字符串的顺序
SUBSTRING()	从字符串的指定位置开始获取指定长度的字符串。与 MID()函数等价
LEFT()	截取并返回字符串的左侧指定个字符
RIGHT()	截取并返回字符串的右侧指定个字符
LPAD()	按照限定长度从左到右截取字符串，当字符串的长度小于限定长度时在左侧填充指定的字符
RPAD()	按照限定长度从左到右截取字符串，当字符串的长度小于限定长度时在右侧填充指定的字符
INSTR()	返回子串在一个字符串中第一次出现的位置。与 LOCATE()和 POSITION(…IN…)函数等价，但参数顺序不同
FIND_IN_SET()	获取子串在含有英文逗号分隔的字符串中的开始位置
LTRIM()	删除字符串左侧的空格，并返回删除后的结果
RTRIM()	删除字符串右侧的空格，并返回删除后的结果
TRIM()	删除字符串左右两侧的空格，并返回删除后的结果
INSERT()	从字符串的指定位置开始使用子串替换指定长度的字符串
REPLACE()	使用指定的子串替换字符串中出现的所有指定字符
CONCAT()	将参数连接成一个新字符串
CONCAT_WS()	使用指定分隔符将参数连接成一个新字符串

表 10-3 中列举了很多字符串函数，读者初次认识时可能并不了解这些函数的使用。下面通过案例演示的方式，以常见的操作为例进行详细讲解。

(1) 获取字符串的长度和字节数。LENGTH()函数用于获取字符串中每个字符占用的字节数，而 CHAR_LENGTH()函数则根据当前 MySQL 的字符集来获取字符串的长度，不同字符集获取的长度不同。例如，在字符集为 gbk 的情况下，获取字符串的长度和字节数。具体 SQL 语句及执行结果如下。

```
mysql> SELECT CHAR_LENGTH('努力的 ren'), LENGTH('努力的 ren');
+-------------------------+---------------------+
| CHAR_LENGTH('努力的 ren')   | LENGTH('努力的 ren')      |
```

```
+---------------------------+----------------------+
|                         6 |                    9 |
+---------------------------+----------------------+
1 row in set (0.00 sec)
```

从上述的执行结果可知，CHAR_LENGTH()函数将中文(多字节字符)算作单个字符进行计算，而 LENGTH()函数在计算时一个中文占用两个字节，因此结果分别为 6 和 9。

(2) 比较两个字符串的大小。STRCMP()函数有两个参数，表示参与比较的字符串。当第 1 个参数大于第 2 个参数时，返回 1；当第 1 个参数等于第 2 个参数时，返回 0；当第 1 个参数小于第 2 个参数时，返回−1。具体 SQL 语句及执行结果如下。

```
mysql> SELECT STRCMP('A','C'), STRCMP('M','D'), STRCMP('12','12');
+------------------+------------------+--------------------+
| STRCMP('A','C')  | STRCMP('M','D')  | STRCMP('12','12')  |
+------------------+------------------+--------------------+
|               -1 |                1 |                  0 |
+------------------+------------------+--------------------+
1 row in set (0.00 sec)
```

值得一提的是，STRCMP()函数是根据参数的校对集设置的比较规则进行比较的，当校对集不兼容时，则必须将其中一个参数的校对集转换为与另一个参数兼容的状态。

(3) 截取字符串。MySQL 中提供的从一个字符串中截取子串的函数有多个，分别为 SUBSTRING()、MID()、LEFT()、RIGHT()、LPAD()和 RPAD()函数。其中，SUBSTRING()与 MID()的功能相同，使用方式也相同，这里仅以 SUBSTRING()为例讲解。关于比较函数的区别如表 10-4 所示。

表 10-4 截取字符串函数比较

函数	截取开始位置	限定长度	填充字符
SUBSTRING()	自定义	自定义	不支持
LEFT()	不支持自定义，只能从左侧开始	自定义	不支持
RIGHT()	不支持自定义，只能从右侧开始	自定义	不支持
LPAD()	不支持自定义，只能从左侧开始	自定义	自定义
RPAD()	不支持自定义，只能从左侧开始	自定义	自定义

小提示： 子串指的是一个字符串 str 中一个或多个字符组成的字符串，它可能是 str 的一部分，也可能与 str 完全相同。

从表 10-4 可知，SUBSTRING()函数适用于自定义截取的开始位置以及截取的长度，LEFT()和 RIGHT 适用于从左侧或右侧截取指定长度的子串，LPAD()和 RPAD()适用于在截取子串时，限定的长度大于字符串长度时根据自定义的字符进行填充占位。

为了读者更好地理解，下面以截取 hello 字符串为例进行演示，具体 SQL 语句及执行结果如下。

```
mysql> SELECT SUBSTRING('hello',2,3) sub,
    -> LEFT('hello',3) l, RIGHT('hello',3) r,
    -> LPAD('hello',6,'*') lpad, RPAD('hello',6,'*') rpad;
+------+------+------+--------+--------+
| sub  | l    | r    | lpad   | rpad   |
+------+------+------+--------+--------+
| ell  | hel  | llo  | *hello | hello* |
+------+------+------+--------+--------+
1 row in set (0.00 sec)
```

在上述 SQL 语句中，函数的第 1 个参数表示待截取子串的字符串；SUBSTRING()函数的第 2 个参数表示截取子串的开始位置，默认字符串首个字符的位置从 1 开始，因此设置为“2”，表示从“hello”的“e”开始截取；SUBSTRING()函数的第 3 个参数、LEFT()、RIGHT()、LPAD()以及 RPAD()函数的第 2 个参数都表示限定截取子串的长度，除 LPAD()和 RPAD()函数外，当限定的长度大于字符串的长度时，则直接截取到字符串的末尾。LPAD()和 RPAD()函数的第 3 个参数表示限定子串的长度大于字符串长度时填充的字符。

另外，SUBSTRING()函数的第 2 个参数还可以是负数，如－1 表示从字符串末尾的字符开始截取。需要注意的是，当设置的截取开始位置超过字符串的长度时，结果将返回空。

(4) 获取子串第一次出现的位置。若要查看子串在字符串中第一次出现的位置，可以使用 INSTR()、LOCATE()或 POSITION(…IN…)函数获取。但当要获取子串在含有英文逗号分隔的字符串中第一次出现的位置时也可以使用 FIND_IN_SET()函数。具体 SQL 语句及执行结果如下。

```
mysql> SELECT INSTR('banana','na') instr,
    -> LOCATE('na','banana') locate,
    -> FIND_IN_SET('in','liner,in,hit,in') find;
+-------+--------+------+
| instr | locate | find |
+-------+--------+------+
|   3   |   3    |  2   |
+-------+--------+------+
1 row in set (0.00 sec)
```

在上述 SQL 语句中，INSTR()函数用于查询第 2 个参数(子串)在第 1 个参数中首次出现的位置，而 LOCATE()和 POSITION(…IN…)函数的参数位置正好与 INSTR()相反。FIND_IN_SET()函数的第 1 个参数表示子串，它必须是字符串中使用逗号分隔的字符(如 liner、in、hit)，否则查询的结果为 0。

(5) 去除字符串两端的空格。在将数据保存到数据库之前，若没有对数据进行任何处理，则数据的两端可能会存在空格，此时可以根据实际的需求去除数据左边、右边或两边的空格。具体 SQL 语句及执行结果如下。

```
mysql> SELECT CONCAT('START', LTRIM('  haha  '), 'END') ltrim,
    -> CONCAT('START', RTRIM('  haha  '), 'END') rtrim,
    -> CONCAT('START', TRIM('  haha  '), 'END') trim;
```

```
+----------------+----------------+---------------+
| ltrim          | rtrim          | trim          |
+----------------+----------------+---------------+
| STARThaha   END | START   hahaEND | STARThahaEND  |
+----------------+----------------+---------------+
1 row in set (0.00 sec)
```

在上述 SQL 语句中，为了直观地看出字符串“ haha ”两端的空格是否也去除，使用 CONCAT()函数将去除空格的字符串拼接在 START 和 END 的中间。因此，从输出结果可知，去掉左端空格后仅 haha 与 END 之间有空格；去掉右端空格后，仅 haha 与 START 之间有空格；当同时去掉两端的空格后，haha 与 START 和 END 之间不再有空格。

(6) 替换子串。当想要替换字符串中所有出现的指定子串时，可以使用 REPLACE()函数完成，但若要替换字符串中指定长度的子串时则可以使用 INSERT()函数实现。具体 SQL 语句及执行结果如下。

```
mysql> SELECT INSERT('welcome', 3, 4, 'HA') first,
    -> INSERT('welcome', -4, 3, 'HA') two,
    -> INSERT('welcome', 7, 10, 'HA') three,
    -> REPLACE('welcome', 'e', 'E') four;
+--------+----------+----------+----------+
| first  | two      | three    | four     |
+--------+----------+----------+----------+
| weHAe  | welcome  | welcomHA | wElcomE  |
+--------+----------+----------+----------+
1 row in set (0.00 sec)
```

在上述 SQL 语句中，INSERT()和 REPLACE()函数的第 1 个参数表示待替换子串的字符串，前者的第 2 个参数为替换的开始字符位置，默认从 1 开始，第 3 个参数用于限定替换子串的最大长度，第 4 个参数用于替换从指定位置开始到限定长度的子串，如将 lco 替换为 HA；后者的第 2 个参数表示替换前字符串中的子串，第 3 个参数指定替换后的内容。

需要注意的是，在使用 INSERT 替换字符串时，开始位置设置为负数(如－4)，则返回值为原字符串。当限定的子串长度大于字符串的长度(如 10)时，仅利用指定的字符串从指定的开始位置替换到字符串的末尾。

4. 日期和时间函数

MySQL 中为了方便日期和时间的处理与转换，提供了丰富的内置函数。其中，常用的如表 10-5 所示。

表 10-5　常用的日期和时间函数

函数名称	作用
CURDATE()	用于获取 MySQL 服务器当前日期，与 CURRENT_DATE()等价
CURTIME()	用于获取 MySQL 服务器当前时间，与 CURRENT_TIME()等价
NOW()	用于获取 MySQL 服务器当前日期和时间，与 LOCALTIME()、CURRENT_TIMESTAMP()和 LOCALTIMESTAMP()都等价

续表

函数名称	作　　用
UNIX_TIMESTAMP()	用于获取 MySQL 服务器当前 UNIX 时间戳
UNIX_TIMESTAMP(data)	将指定日期 data 时间转换为 UNIX 时间戳并返回
DATEDIFF()	判断两个日期之间的天数差距,参数日期必须使用字符串格式
DATE()	获取日期或日期时间表达式中的日期部分
TIME()	获取指定日期事件表达式中的时间部分
WEEK()	返回指定日期的周数
DAYNAME()	返回日期对应的星期名称(英文全称)
DAYOFMONTH()	返回指定日期中的天(1～31),与 DAY()等价
DAYOFYEAR()	返回指定日期的天数
DAYOFWEEK()	返回日期对应的星期几(1＝周日,2＝周一,……,7＝周六)
FROM_UNIXTIME()	将指定的时间戳转成对应的日期时间格式
EXTRACT()	按照指定参数提取日期中的数据
DATE_SUB()	从日期中减去时间值(时间间隔)
DATE_ADD()	在指定的日期上添加日期时间

在表 10-5 中,所有函数的返回值均与时区设置有关。另外,UNIX 时间戳(Unix timestamp)是一种时间的表示方式,定义了从格林尼治时间 1970 年 01 月 01 日 00 时 00 分 00 秒起至现在的总秒数,以 32 位二进制数表示。其中,1970 年 01 月 01 日零点也叫作 Unix 纪元。

小提示:

(1) 时区是地球上各区域使用的同一个时间定义方式,全球共分为 24 个时区,例如中国采用的是东八区的时间。

(2) 默认情况下,MySQL 服务器的时区与当前系统的时区相同,读者可利用 SELECT 查看系统变量 time_zone 的值为 SYSTEM。

(3) 在管理数据库时,若要临时修改 MySQL 服务器的时区,可以修改系统变量 time_zone 的值。而若要永久修改时区,则可以在 MySQL 的配置文件中添加时区的设置。

```
#方式 1:临时修改,设置系统的临时变量
SET time_zone = '+8:00';
#方式 2:打开 my.ini,添加以下配置
default-time_zone = '+8:00'
```

在上述的修改方式中,为 MySQL 的 my.ini 文件添加配置项后,需要重新启动服务器,配置才能生效。

为了读者更好地理解,下面以案例的方式演示 MySQL 中日期与时间函数的使用。具体如下。

(1) 获取更精确的服务器时间。MySQL 提供的 NOW()、LOCALTIME()、

CURRENT_TIMESTAMP()、LOCALTIMESTAMP()都可以获取当前服务器的时间，它们的使用方式相同。下面以 NOW()为例，具体 SQL 语句及执行结果如下。

```
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2018-06-07 16:12:50 |
+---------------------+
1 row in set (0.00 sec)
```

从上述的执行结果可知，默认情况下 NOW()获取的日期时间格式为"YYYY-MM-DD HH:MM:SS"。除此之外，还可以为函数传递参数([0,6]之间的任意数字)，设置获取时间的毫秒位数，从而可以获取更加精确的时间信息。例如，获取保留 4 位数字的毫秒时间，具体 SQL 语句及执行结果如下。

```
mysql> SELECT NOW(4);
+--------------------------+
| NOW(4)                   |
+--------------------------+
| 2018-06-07 16:56:57.7627 |
+--------------------------+
1 row in set (0.00 sec)
```

在上述执行结果中，在时间后的小数部分".7627"就是获取的毫秒数，它的位数由 NOW()函数的参数值决定，最小值为 0，最大值为 6。

除此之外，MySQL 提供的 SYSDATE()函数与 NOW()函数一样也可以获取当前服务器的日期时间，区别在于前者获取的是动态的实时时间，后者获取的是语句开始执行的时间。对比示例及执行结果如下所示。

```
mysql> SELECT NOW(), SYSDATE(), SLEEP(2), NOW(), SYSDATE() \G
*************************** 1. row ***************************
    NOW(): 2018-06-07 17:09:34
SYSDATE(): 2018-06-07 17:09:34
  SLEEP(2): 0
    NOW(): 2018-06-07 17:09:34
SYSDATE(): 2018-06-07 17:09:36
1 row in set (2.00 sec)
```

上述 SQL 语句中，利用 SLEEP(2)延迟 2 秒，然后在延迟前后分别使用 NOW()和 SYSDATE()获取日期时间。通过执行结果对比可知，SYSDATE()获取的时间前后正好差 2 秒，而 NOW()获取的前后值相同，即 SELECT 语句开始执行的时间。

(2) 格式化日期。在实际开发中，数据库存储的日期时间经常是 UNIX 时间戳，但是对于用户来说，时间戳的直接输出，会让其看到一个毫无意义的整型数值。为了将时间戳表示的时间以友好的形式显示出来，可以对时间戳进行格式化。MySQL 中提供了 FROM_UNIXTIME()函数实现自定义格式化后日期的显示样式，常用的格式字符如表 10-6 所示。

表 10-6 常用格式字符

分类	格式字符	说明
年	%Y	4 位数字表示的完整年份，如 1949、2018
	%y	2 位数字表示的年份，如 99、03
月	%m	两位数字表示的月份，有前导零，返回值 00～12
	%c	数字表示的月份，无前导零，返回值 0～12
	%M	月份，完整的文本格式，如 January、March
日	%d	月份中的第几天，有前导零，返回值 00～31
	%e	月份中的第几天，无前导零，返回值 0～31
时间	%h 或 %I	小时，12 小时格式，有前导零，返回值 01～12
	%H	小时，24 小时格式，有前导零，返回值 00～23
	%i	有前导零的分钟数，返回值 00～59
	%s 或 %S	有前导零的秒数，返回值 00～59
星期	%w	星期几，返回值 0（表示星期日）～6（表示星期六）
	%W	星期几，完整的文本格式，如 Sunday、Saturday

在表 10-6 中，格式字符前的“%”是必需的。为了读者更好地理解，下面以一个案例演示如何对时间戳进行格式化。具体 SQL 语句及执行结果如下。

```
mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(), '%W') '星期',
    -> FROM_UNIXTIME(UNIX_TIMESTAMP(), '%y/%c/%e') '日期',
    -> FROM_UNIXTIME(UNIX_TIMESTAMP(), '%H:%i:%s') '时间';
+----------+--------+----------+
| 星期      | 日期    | 时间      |
+----------+--------+----------+
| Thursday | 18/6/7 | 18:29:16 |
+----------+--------+----------+
1 row in set (0.00 sec)
```

在上述 SQL 语句中，UNIX_TIMESTAMP()用于获取 MySQL 服务器当前 UNIX 时间戳，然后利用 FROM_UNIXTIME()根据不同的格式字符串获取星期、日期和时间，且格式字符之间可以随意设置分隔符进行连接。如以上的日期使用“/”分隔年月日，时间使用“:”分隔时分秒等。

（3）提取日期时间。MySQL 为了方便获取更加精确的数据，提供了 EXTRACT()函数从指定的日期中获取想要的日期时间，如当前日期的季度、年份、时分等。EXTRACT()可以提取的日期时间类型如表 10-7 所示。

表 10-7 日期时间类型

MICROSECOND	SECOND	MINUTE
HOUR	DAY	WEEK

续表

MONTH	QUARTER	YEAR
SECOND_MICROSECOND	MINUTE_MICROSECOND	MINUTE_SECOND
HOUR_MINUTE	DAY_SECOND	DAY_MINUTE
DAY_HOUR	YEAR_MONTH	

在表 10-7 中，带下划线“_”的类型获取的内容是两个单词的组合，例如 MINUTE_SECOND 获取的是“分秒”的组合。另外，以“DAY_”开头的类型获取的内容不包含日期(如 8 号)，而是从“小时”开始到符号“_”后指定的类型，如 DAY_MINUTE 获取的是“时分”的组合与 HOUR_MINUTE 获取的内容相同。

为了读者更好地理解，下面以获取当前日期的时分秒为例进行演示，具体 SQL 语句及执行结果如下。

```
mysql> SELECT EXTRACT(DAY_SECOND FROM NOW());
+----------------------------------+
| EXTRACT(DAY_SECOND FROM NOW())   |
+----------------------------------+
|                        183509    |
+----------------------------------+
1 row in set (0.00 sec)
```

在上述执行结果中，从 NOW()获取的日期中提取的时间是一个由时分秒组成的一串数字“183509”。其中，“18”表示小时，“35”表示分钟，“09”表示秒。

(4) 添加或减少日期时间。生活中食品都是有过期时间的，此时在项目中存储此类商品信息时，就可以直接采用 MySQL 提供的添加或减少日期时间的方式实现。通常情况下，使用以下两种方式实现。基本语法格式如下。

```
#方式 1:直接加或减
日期时间 +/- INTERVAL 表达式 日期时间类型
#方式 2:使用函数
DATE_ADD(日期时间, INTERVAL 表达式 日期时间类型)
DATE_SUB(日期时间, INTERVAL 表达式 日期时间类型)
```

在上述语法中，日期时间是一个字符串，INTERVAL 是指定添加或减少的日期时间的关键字，表达式推荐使用“YYYY-MM-DD HH：MM：SS”的形式设置，日期时间类型请参考表 10-7。

为了读者更好地理解，下面通过案例演示以上两种方式如何实现添加或减少日期时间。具体 SQL 语句及执行结果如下。

```
mysql> SELECT '2018-06-07 18:48:07'+INTERVAL 1 DAY  one,
    -> '2018-06-07 18:48:07' - INTERVAL '2 10:02:00' DAY_SECOND two,
    -> DATE_ADD('2018-06-07 18:48:07', INTERVAL '4' YEAR) three,
    -> DATE_SUB('2018-06-07 18:48:07', INTERVAL '20' MINUTE) four \G
*************************** 1. row ****************************
```

```
      one: 2018-06-08 18:48:07
      two: 2018-06-05 08:46:07
    three: 2022-06-07 18:48:07
     four: 2018-06-07 18:28:07
1 row in set (0.00 sec)
```

5. 加密和散列函数

MySQL 提供的加密函数主要用于对数据进行加密，相对明文存储，经过算法计算后的字符串不会被人直接看出保存的是什么数据，在一定程度上能够有效保证数据的安全。散列函数又称为哈希(Hash)函数，用于通过散列算法计算数据的散列值。常见的加密和散列函数如表 10-8 所示。

表 10-8　加密和散列函数

函 数 名 称	作　　用
MD5()	使用 MD5 计算并返回一个 32 位的散列字符串
AES_ENCRYPT()	使用密钥对字符串进行加密，默认返回一个 128 位的二进制数
AES_DECRYPT()	使用密钥对密码进行解密
SHA1()或 SHA()	利用安全散列算法 SHA-1 计算字符串，返回 40 个十六进制数字组成的字符串
SHA2()	利用安全散列算法 SHA-2 计算字符串
ENCODE()	使用密钥对字符串进行编码，默认返回一个二进制数
DECODE()	使用密钥对密码进行解码
PASSWORD()	计算并返回一个 41 位的密码字符串

表 10-8 中，加密函数 ENCODE()、DECODE()从 MySQL 5.7 以及 PASSWORD()函数从 5.7.6 都开始不推荐使用，并且将在之后的 MySQL 版本中被删除。因此，请读者慎重选择。

为了读者更好地理解，下面以 MD5()、SHA()和 SHA2()函数为例演示散列函数的使用。具体 SQL 语句及执行结果如下。

```
mysql> SELECT MD5('123456'),
    -> SHA1('123456'),
    -> SHA2('123456',256) \G
*************************** 1. row ***************************
     MD5('123456'): e10adc3949ba59abbe56e057f20f883e
    SHA1('123456'): 7c4a8d09ca3762af61e59520943dc26494f8941b
SHA2('123456',256): 8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca120
                    20c923adc6c92
1 row in set (0.00 sec)
```

在上述 SQL 语句中，MD5()返回的是 32 个十六进制数字组成的字符串；SHA1()也可称为 SHA()，它采用的是 160 位长度的安全散列算法，返回的是 40 个十六进制数字组成的字符串；而 SHA2()函数需要通过第 2 个参数设置采用多少位长度的安全散列算法，它的可选值为 224、256、384、512 或 0(相当于 256)。另外，当参数为 NULL 时，它们的返回值均为 NULL。

小提示：MD5()和 SHA()函数经常用于密码的非明文存储，也就是计算正确密码的散列值后保存，然后计算给定密码的散列值，判断两个散列值是否相同。对于敏感信息，建议在客户端就考虑使用加密密钥进行加密，在传输时建议尽量采用 SSL 连接，从多方位的角度尽量保证数据的安全。

6. 系统信息函数

为了方便查看 MySQL 服务器的系统信息，如 MySQL 版本号、登录服务器的用户名、主机地址等。常见的系统信息函数如表 10-9 所示。

表 10-9 系统信息函数

函数名称	作用
VERSION()	用于获取当前 MySQL 服务实例使用的 MySQL 版本号
DATABASE()	用于获取当前操作的数据库，与 SCHEMA()函数等价
USER()	用于获取登录服务器的主机地址及用户名，与 SYSTEM_USER()和 SESSION_USER()函数等价
CURRENT_USER()	用于获取该账户名允许通过哪些登录主机连接 MySQL 服务器
CONNECTION_ID()	用于获取当前 MySQL 服务器的连接 ID
BENCHMARK()	重复执行一个表达式
LAST_INSERT_ID()	获取当前会话中最后一个插入的 AUTO_INCREMENT 列的值

在表 10-9 中，BENCHMARK()函数经常用于检测标量表达式的运行性能，这对于优化 MySQL 数据库具有重要的意义。具体使用的 SQL 语句及执行结果如下。

```
mysql> SELECT BENCHMARK(10,'3*7');
+---------------------+
| BENCHMARK(10,'3*7')          |
+---------------------+
|                         0          |
+---------------------+
1 row in set (0.00 sec)
```

在上述 SQL 语句中，BENCHMARK()的第 1 个参数表示重复执行的次数，第 2 个参数表示标量表达式或标量子查询。该函数执行后的返回值始终为 0。因此，通常根据此函数的执行时间判断运行性能。具体 SQL 语句及执行结果如下。

```
mysql> use shop;
Database changed
mysql> SELECT BENCHMARK(1000000,'SELECT name FROM sh_goods WHERE id=2');
+-----------------------------------------------------------+
|BENCHMARK(1000000,'SELECT name FROM sh_goods WHERE id=2')                    |
+-----------------------------------------------------------+
|                                                                0           |
+-----------------------------------------------------------+
1 row in set (0.01 sec)
```

7. JSON 函数

MySQL 中为了方便对 JSON 类型的数据进行操作，提供了很多关于 JSON 的函数。例如，创建 JSON 数组和对象、获取 JSON 中的数据，删除 JSON 中的数据等操作。常见的函数如表 10-10 所示。

表 10-10 常用的 JSON 函数

函数名称	描述
JSON_ARRAY()	创建 JSON 数组
JSON_OBJECT()	创建 JSON 对象
JSON_CONTAINS()	JSON 文档中是否包含路径中指定对象
JSON_CONTAINS_PATH()	JSON 文档中是否包含路径中的任意数据
JSON_EXTRACT()	从 JSON 文档返回数据
JSON_KEYS()	从 JSON 文档中获取数组中的键
JSON_SEARCH()	获取 JSON 文档中值的路径
JSON_ARRAY_APPEND()	将数据追加到 JSON 文档的指定路径中
JSON_ARRAY_INSERT()	将数据插入到 JSON 数组指定路径前
JSON_DEPTH()	获取 JSON 文档的最大深度
JSON_INSERT()	将数据插入 JSON 文档
JSON_LENGTH()	获取 JSON 文档中元素的数量
JSON_MERGE_PATCH()	合并 JSON 文档，替换重复键的值
JSON_MERGE_PRESERVE()	合并 JSON 文档，保留重复的键
JSON_PRETTY()	以友好的格式打印 JSON 文档
JSON_REMOVE()	从 JSON 文档中删除数据
JSON_REPLACE()	替换 JSON 文档中的值
JSON_SET()	向 JSON 文档中插入数据
JSON_TYPE()	获取 JSON 值的类型
JSON_VALID()	JSON 值是否有效

为了读者更好地理解，接下来通过案例演示 JSON 函数常见的用法。具体如下。

(1) 创建 JSON 数组和对象。MySQL 提供了两种方式将给定的值或键值对转换成 JSON 数组或 JSON 对象，基本语法格式如下。

```
JSON_ARRAY([val[,val]…])                    #创建 JSON 数组
JSON_OBJECT([key,val[,key,val]…])           #创建 JSON 对象
```

在上述语法中，JSON_ARRAY () 函数的参数 val 表示 JSON 数组的元素值，函数 JSON_OBJECT()的参数 key 和 val 必须成对出现表示 JSON 对象的键和值，所以当其参数

个数为奇数或 key 的值为 NULL 时，程序都会发生错误。具体 SQL 语句及执行结果如下。

```
mysql>SELECT JSON_ARRAY('cake',2,NULL),
    ->JSON_OBJECT("id",12,"name","Tom");
+---------------------------+-----------------------------------+
| JSON_ARRAY('cake',2,NULL)  | JSON_OBJECT("id",12,"name","Tom")   |
+---------------------------+-----------------------------------+
| ["cake", 2, null]          | {"id": 12, "name": "Tom"}           |
+---------------------------+-----------------------------------+
1 row in set (0.00 sec)
```

(2) 添加 JSON 数据。对于已添加了 JSON 数据的字段来说，若想要继续添加 JSON 数据或替换已有的值时，可以使用 MySQL 提供的 JSON_SET()、JSON_INSERT()或 JSON_REPLACE()函数实现。这 3 个函数在使用时有以下几点的区别。

- JSON_SET()函数添加的 JSON 数据不存在时，则添加，否则替换现有值。
- JSON_INSERT()函数添加的 JSON 数据不存在时，则插入，否则保留现有值。
- JSON_INSERT()函数添加的 JSON 只替换现有的值。

下面以 JSON_SET()为例进行演示，具体 SQL 语句及执行结果如下。

```
mysql>SELECT JSON_SET('{"id": 12}', "$.email", "test@163.com") c1,
    ->JSON_SET('{"id": 12}', "$.id", "8") c2 \G
*************************** 1. row ***************************
c1: {"id": 12, "email": "test@163.com"}
c2: {"id": "8"}
1 row in set (0.00 sec)
```

从以上的执行结果可知，JSON_SET()添加的 email 在{"id": 12}中不存在，则添加，此时 JSON 对象含有两个键值对；当 JSON_SET()添加的 id 在{"id": 12}中已存在时，则利用 8 替换原来的 12，所以最后的 JSON 对象中 id 键的值为 8。

以上示例中，“$.email”是获取 JSON 数据的路径方式，“$”表示 JSON 对象{"id": 12}，$.email 用于获取 JSON 对象中键名为 email 的对应值。

(3) 删除 JSON 数据。当 JSON 类型中保存的部分数据已没有意义时，可以使用 MySQL 提供的 JSON_REMOVE()函数将其删除。具体 SQL 语句及执行结果如下。

```
mysql>SELECT JSON_REMOVE('["a",[2,3],"b"]',"$[1]");
+----------------------------------------+
| JSON_REMOVE('["a",[2,3],"b"]',"$[1]")   |
+----------------------------------------+
|                         ["a", "b"]     |
+----------------------------------------+
1 row in set (0.00 sec)
```

在上述 SQL 语句中，JSON_REMOVE()的第 1 个参数表示 JSON 数据，第 2 个参数表示待删除的 JSON 数据路径。

(4) 合并 JSON 数据。在数据库操作时若需要将多个 JSON 数据合并可以使用 JSON_MERGE_PATCH()和 JSON_MERGE_PRESERVE()函数，前者在合并数据时若有相同键

的数据，则后面的数据会覆盖前面的数据；而后者会保留所有的数据。具体 SQL 语句及执行结果如下。

```
mysql> SELECT JSON_MERGE_PATCH('{"a":1,"b":2}',
    -> '{"a":["one",11],"b":44,"c":3}') PATCH,
    -> JSON_MERGE_PRESERVE('{"a":1,"b":2}',
    -> '{"a":["one",11],"b":44,"c":3}') PRESERVE \G
*************************** 1. row ***************************
   PATCH: {"a": ["one", 11], "b": 44, "c": 3}
PRESERVE: {"a": [1, "one", 11], "b": [2, 44], "c": 3}
1 row in set (0.00 sec)
```

在上述 SQL 语句中，JSON_MERGE_PATCH()函数在合并 JSON 对象时，第 2 个参数对象中名为 a 和 b 的键覆盖了第 1 个参数同名的键值，而 JSON_MERGE_PRESERVE()函数会保留所有的键值。

(5) 搜索 JSON 值和路径。MySQL 提供了 JSON_SEARCH()函数用于根据 JSON 值获取其对应的路径，JSON_EXTRACT()函数根据 JSON 路径获取对应的 JSON 值。具体 SQL 语句及执行结果如下。

```
mysql> SELECT
    -> JSON_SEARCH('["cookie","23",{"eat":"cookie"}]','all',"cookie") c1,
    -> JSON_EXTRACT('["cookie","23",{"eat":"cookie"}]','$[1]','$[0]') c2
    -> \G
*************************** 1. row ***************************
c1: ["$[0]", "$[2].eat"]
c2: ["23", "cookie"]
1 row in set (0.00 sec)
```

在上述 SQL 语句中，JSON_SEARCH()函数的第 2 个参数的可选值为 all 或 one，前者表示获取所有 JSON 值对应的路径，后者表示获取到一个 JSON 值对应的路径就停止匹配。JSON_EXTRACT()函数的第 2 个和第 3 个参数表示待提取值的 JSON 路径，使用时可以设置多个，多个之间使用逗号分隔即可。

值得一提的是，在 MySQL 5.7.9 和更高版本中，提取数据表中的 JSON 字段数据时，可以简写 JSON_EXTRACT()函数，具体对比如下。

```
#提取字段中的 JSON 值
JSON_EXTRACT(字段名,JSON 路径)                          #等价于  字段名->JSON 路径
#提取字段中的 JSON 值,并删除值的引号
JSON_UNQUOTE( JSON_EXTRACT(字段名,JSON 路径) )          #等价于  字段名->>JSON 路径
```

从上述对比可知，当获取数据表中 JSON 字段的某个数据时，可以使用“->”作为 JSON_EXTRACT()函数的别名；而“->>”不仅能满足“->”的功能，还可以删除获取到的 JSON 值的引号。

8. 其他常用函数

除了以上讲解的 7 种常用函数外，MySQL 中还提供了一些非常有用的函数，例如 IP

地址与数字转换的函数等。具体如下。

(1) 转换 IP 地址与数字。在项目开发时为了节省存储 IP 地址的空间,提升开发的效率,推荐使用 MySQL 提供的 INET_ATON()函数将 IP 地址转换为数字存储在对应的数据表的字段中,在使用时再利用 INET_NTOA()函数将数值转换为 IP 地址。具体 SQL 语句及执行结果如下。

```
mysql> SELECT INET_ATON('192.168.22.11');
+----------------------------+
| INET_ATON('192.168.22.11') |
+----------------------------+
|                 3232241163 |
+----------------------------+
1 row in set (0.00 sec)
```

在上述 SQL 语句中,INET_ATON()函数会将参数 IP 地址转换为对应的数字,计算的方式为 $11+22*\mathbf{256}+168*\mathbf{256^2}+192*\mathbf{256^3}$。需要注意的是,传入的非法的 IP 地址(如 1.1.1.1.1)系统将返回 NULL。

另外,在需要将数值转换为 IP 地址时,只需调用 INET_ATON()函数将数字转换为对应的 IP 地址即可,具体 SQL 语句及执行结果如下。

```
mysql> SELECT INET_NTOA(3232241163);
+-----------------------+
| INET_NTOA(3232241163) |
+-----------------------+
| 192.168.22.11         |
+-----------------------+
1 row in set (0.00 sec)
```

(2) 延迟语句执行的时间。在操作和管理 MySQL 时,若需要延迟语句执行的时间,则可以使用 MySQL 提供的 SLEEP()函数,具体 SQL 语句及执行结果如下。

```
mysql> SELECT SLEEP(2);
+----------+
| SLEEP(2) |
+----------+
|        0 |
+----------+
1 row in set (2.00 sec)
```

在上述 SQL 语句中,SLEEP()函数的参数是延迟的秒数。从执行结果的执行时间即可以看出延迟的时间。

(3) 获取唯一标识符。在 MySQL 中除了在创建表时设置 AUTO_INCREMENT 自增序列实现唯一值外,还可以利用 MySQL 提供的 UUID()函数在同一时间同一空间实现唯一的标识符。

```
mysql> SELECT UUID();
+--------------------------------------+
```

```
| UUID()                               |
+--------------------------------------+
| 7e0072e3-6bb4-11e8-9ebb-40167e6696ec |
+--------------------------------------+
1 row in set (0.00 sec)
```

在上述执行结果中，UUID()获取的标识符由 32 位小写的十六进制数字组成，共分为 5 部分，前 3 部分数字根据时间戳的低中高部分生成，高位部分还包括 UUID 版本号，第 4 部分用于保存时间的唯一性，第 5 部分用于保存空间的唯一性。需要注意的是，理论上尽管 UUID()获取的是唯一标识符，但也不能保证会有其他方式生成同样的标识符。

10.1.2　自定义函数

MySQL 中除了提供丰富的内置函数外，还支持用户自定义函数，用于实现某种功能。它是由多条语句组成的语句块，每条语句都是一个符合语句定义规范的个体，需要语句结束符——分号(;)，而 MySQL 一旦遇见语句结束符就会自动开始执行，但函数是一个整体，只有在被调用时才会被执行，那么在定义函数时就需要临时修改语句结束符。基于语法格式如下。

```
DELIMITER 新结束符号
  自定义函数
新结束符号
DELIMITER ;
```

在上述语法中，自定义的新结束符号推荐使用系统非内置的符号，如$$。在使用 DELIMITER 修改语句结束符后，自定义函数中就可以正常使用分号结束符，系统由于不再将分号作为语句结束符，从而不会自动执行自定义函数的 SQL 语句。

需要注意的就是完成函数的自定义后，首先需要使用新结束符号进行结束，然后需使用 DELIMITER 将语句结束符改回原来的分号(;)。

1. 自定义函数语法

在了解了如何修改语句结束符后，接下来看一下 MySQL 中如何自定义函数，基本语法格式如下。

```
CREATE FUNCTION 函数名 ([参数名 数据类型, …]) RETURNS 返回值类型
[BEGIN]
  #函数体
  RETURN 返回值数据; #数据必须与结构中定义的返回值类型一致
[END]
```

从上述的语法可知，自定义函数包括 FUNCTION 关键字、函数名、参数、返回值类型以及函数体和返回值。其中，函数名的命名必须符合 MySQL 的语法规定，推荐使用字母、数字和下划线；可选参数都是由一个参数名称(不区分大小写)和数据类型组成，它们之间使用空格分隔，多个参数之间使用逗号分隔，即使没有可选参数，在定义函数时，函数名也必须跟上一个空的小括号()；当函数有返回值时，若其数据类型与指定的类型不相同时，会进行自

动类型转换。

另外，自定义函数体内含有多条语句时，必须使用复合语句语法 BEGIN…END 包裹函数体，该语法以 BEGIN 开头，以 END 结尾，在这两个关键字中间可以包含零条或多条 SQL 语句，经常被应用在自定义函数、存储过程、触发器或事件中。不过仅有一条语句时，可以省略 BEGIN…END，但通常情况下推荐用 BEGIN…END 包裹。

默认情况下，自定义的函数与默认选择的数据库相关联，用于表示此函数只有在这个数据库中才能使用，若没有选择数据库就直接自定义函数，程序会给出相关的提示信息。

为了读者更好地理解，下面在 shop 数据库中，自定义一个对指定用户打招呼的函数，具体 SQL 语句及执行结果如下。

```
mysql> DELIMITER $$
mysql> CREATE FUNCTION sayHello(name VARCHAR(30)) RETURNS VARCHAR(50)
    -> BEGIN
    ->   RETURN CONCAT('Hello ', name, '!');
    -> END
    -> $$
Query OK, 0 rows affected (0.00 sec)
mysql> DELIMITER ;
```

在上述 SQL 语句中，sayHello 是自定义函数的名称，name 是自定义函数的形式参数，其后是该参数的数据类型，RETURNS 指定返回值的类型，BEGIN 和 END 之间定义的是函数体。

值得一提的是，自定义函数需要规范返回值的类型，那么在函数内部则不能使用 SELECT 指令获取结果集，若需要相关的查询结果只能将其保存到指定的变量中，此处读者了解即可，具体关于变量的使用将在 10.3 节中讲解。

2. 查看函数

在自定义函数完成后，可以通过 SHOW 查看函数的创建语句，具体 SQL 语句及执行结果如下。

```
mysql> SHOW CREATE FUNCTION sayHello \G
*************************** 1. row ***************************
            Function: sayHello
            sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_
FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
     Create Function: CREATE DEFINER=`root`@`localhost` FUNCTION `sayHello`(name varchar(30)) RETURNS
varchar(50) CHARSET latin1
BEGIN
        RETURN CONCAT('Hello',name);
END
character_set_client: gbk
collation_connection: gbk_chinese_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

上述执行结果包括创建此函数的用户名以及主机地址，SQL 语句以及相关的 SQL 模

式、字符集和校对集。其中，自定义函数时可以利用 DEFINER 指定创建的用户，省略时服务器自动将当前登录的用户作为函数的定义者，如这里使用 root 用户。

除此之外，还可以查看自定义函数的状态以及系统中所有的自定义函数，具体 SQL 语句及执行结果如下。

```
mysql> SHOW FUNCTION STATUS LIKE 'sayHello' \G
*************************** 1. row ***************************
                  Db: shop
                Name: sayHello
                Type: FUNCTION
             Definer: root@localhost
            Modified: 2018-06-11 10:41:29
             Created: 2018-06-11 10:41:29
       Security_type: DEFINER
             Comment:
character_set_client: gbk
collation_connection: gbk_chinese_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

在上述查询 SQL 语句中，LIKE 后指定的是匹配模式，表示获取 sayHello 函数的状态，省略 LIKE 匹配模式时则可查看所有函数的状态。从执行结果可看到函数的所属数据库、名称、类型、定义者、修改和创建时间等信息。

小提示： MySQL 中自定义函数、存储过程、触发器以及事件等的操作用户必须具备相关的权限，如自定义函数的用户必须有 CREATE ROUTINE 权限，其他操作对应的权限请参考本书第 7 章的表 7-7。

3. 调用函数

函数定义完成后，若想要它在程序中发挥作用，需要调用才能使其生效。它的使用与 MySQL 中内置函数的调用相同。基本语法格式如下。

```
SELECT 函数名 1(实参列表), 函数名 2(实参列表), …;
```

在上述语法中，调用函数时，需要指定每个函数所属的数据库，其使用的形式(数据库.函数名)与选择数据库中的数据表相同。调用函数的实参列表指的是为定义函数时设置的形参传递具体的值，如形参为字符串型的 name，则调用函数时可以传递同类型的任意数据，如 TOM，JIMMY。

为了读者更好地理解，下面以调用 shop 数据库下的 sayHello 函数为例进行演示。具体 SQL 语句及执行结果如下。

```
mysql> SELECT sayHello('TOM');
+-----------------+
| sayHello('TOM') |
+-----------------+
| Hello TOM!      |
+-----------------+
1 row in set (0.00 sec)
```

4. 删除函数

MySQL为删除数据库中函数提供了专门的SQL语句，基本语法格式如下。

```
DROP FUNCTION [IF EXISTS] 函数名;
```

例如，删除shop数据库下的sayHello函数，具体SQL语句及执行结果如下。

```
mysql>DROP FUNCTION [IF EXISTS] sayHello;
Query OK, 0 rows affected (0.00 sec)
```

10.2 存储过程

在数据库管理中，使用到的程序除了函数外，还有存储过程可以在数据库中进行一系列复杂的操作。在使用时只需将复杂的SQL语句集合封装成一个代码块，就可以重复使用，减少数据库管理人员的工作量。本节将针对创建存储过程进行详细讲解。

10.2.1 存储过程的概念

对于SQL编程而言，存储过程是数据中的一个重要的对象，它是在大型数据库系统中一组为了完成特定功能的SQL语句集，在第一次使用经过编译后，再次调用就不需要重复编译，因此执行效率比较高。它与函数在数据库中的异同点如下。

（1）存储过程与函数的相同点在于，它们的目的都是为了可重复地执行数据库SQL语句的集合，并且都是经过一次编译后，后面再次需要时直接执行即可。

（2）存储过程与函数的不相同点有4个，具体如下。

- 语法中实现的标识符不同，存储过程使用PROCEDURE，函数为FUNCTION。
- 存储过程在创建时没有返回值，而函数在定义时必须设置返回值。
- 存储过程没有返回值类型，且不能将结果直接赋值给变量；而函数定义时需要设置返回值类型，且在调用时必须将返回值赋给变量。
- 存储过程必须通过CALL进行调用，不能使用SELECT调用；而函数则可在SELECT语句中直接使用。

10.2.2 存储过程的创建与执行

1. 创建存储过程

存储过程在创建时与创建函数相同，首先都需要临时修改语句结束符号。然后再利用CREATE语句进行创建，其基本语法格式如下。

```
DELIMITER 新结束符号
CREATE PROCEDURE 过程名字 ([[ IN | OUT | INOUT ] 参数名称 参数类型])
BEGIN
  过程体
END
```

```
新结束符号
DELIMITER ;
```

在上述语法中，创建存储过程的关键字为 PROCEDURE，在为存储过程设置参数时，在参数名前还可以指定参数的来源及用途，可选值分别为 IN(默认值)、OUT 和 INOUT。三者的区别如下所示。

- IN：表示输入参数，即参数是在调用存储过程时传入到存储过程里面使用，传入的数据可以是直接数据(如 5)，也可以是保存数据的变量。
- OUT：表示输出参数，初始值为 NULL，它是将存储过程中的值保存到 OUT 指定的参数中，返回给调用者。
- INOUT：表示输入输出参数，即参数在调用时传入到存储过程，同时在存储过程中操作之后，又可将数据返回为调用者。

关于变量的内容，此处读者了解即可，具体的内容会在下一节中详细讲解。

为了读者更好地理解，下面通过一个案例演示在 shop 数据库中创建存储过程，具体 SQL 语句及执行结果如下。

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE proc(IN sid INT)
    -> BEGIN
    ->     SELECT id, name FROM sh_goods_category where id > sid;
    -> END
    -> $$
Query OK, 0 rows affected (0.00 sec)
mysql> DELIMITER ;
```

在上述语句中，sid 表示调用存储过程时传递进来的参数，根据此参数在存储过程体内从 sh_goods_category 表中获取 id 大于此值的数据。值得一提的是，创建存储过程与自定义函数相同，都需要指定与其相关联的数据库，若默认没有选择数据库，则程序会给出相关的提示信息。

2. 查看存储过程

创建完存储过程后，可以使用 MySQL 专门提供的语句查看存储过程，实现方式与查看函数的相同，区别在于实现的关键字不同。其基本语法格式如下。

```
#查看存储过程的创建语句
SHOW CREATE PROCEDURE 过程名;
#根据指定的模式查看所有符合要求的存储过程
SHOW PROCEDURE STATUS [LIKE 匹配模式];
```

为了读者更好地理解，下面以查看 shop 数据库下的 proc 存储过程的状态为例进行演示，具体 SQL 语句及执行结果如下。

```
mysql> SHOW PROCEDURE STATUS LIKE 'proc'\G
*************************** 1. row ***************************
```

```
                    Db: shop
                  Name: proc
                  Type: PROCEDURE
               Definer: root@localhost
              Modified: 2018-06-11 16:47:27
               Created: 2018-06-11 16:47:27
         Security_type: DEFINER
               Comment:
  character_set_client: gbk
  collation_connection: gbk_chinese_ci
    Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

从上述执行结果可看出，存储过程的状态信息与自定义函数的相同，区别在于自定义函数的 Type 为值 FUNCTION，而存储过程的对应值为 PROCEDURE。

3. 调用存储过程

要想存储过程发挥作用，必须使用 MySQL 提供的 CALL 语句调用。另外，由于存储过程和数据库相关，如果要执行其他数据库中的存储过程，则调用时需要指定数据库名称。基本语法格式如下。

```
CALL 数据库名.存储过程名称([实参列表]);
```

在上述语法中，实参列表传递的参数需要与创建存储过程的形参相对应，当形参被指定为 IN 时，则实参值可以为变量或是直接数据；当形参被指定为 OUT 或 INOUT 时，调用存储过程传递的参数必须是一个变量，用于接收返回给调用者的数据。

例如，调用 shop 数据库下的 proc 存储过程，获取 sh_goods_category 表中 id 大于指定值的记录，具体 SQL 语句及执行结果如下。

```
mysql>CALL proc(14);
+----+------+
| id | name |
+----+------+
| 15 | 风衣 |
| 16 | 毛衣 |
+----+------+
2 rows in set (0.00 sec)
Query OK, 0 rows affected (0.01 sec)
```

上述执行结果有两行描述的信息，“2 rows in set (0.00 sec)”表示执行存储过程体内的 SQL 语句的描述信息；“Query OK, 0 rows affected (0.01 sec)”表示调用存储过程的结果描述信息。

10.2.3 存储过程的修改与删除

在实际数据库管理中，业务需求更改的情况时有发生，这样就不可避免地需要修改存储过程的特性。在 MySQL 中可以使用 ALTER 语句修改存储过程的特性，其基本语法格式

如下。

```
ALTER PROCEDURE 存储过程名称 [特征]
```

上述语法中的特征指的是存储过程需要修改哪个部分。具体如表 10-11 所示。需要注意的是，ALTER PROCEDURE 不能更改存储过程的参数或主体。

表 10-11　存储过程特征选项

特征选项	描述
COMMENT '注释内容'	为存储过程设置注释信息
LANGUAGE SQL	存储过程体是 SQL 语句，未来可能会支持其他类型语句
CONTAINS SQL	表示子程序中包含除读或写数据的 SQL 语句
NO SQL	表示子程序中不包含 SQL 语句
READS SQL DATA	表示子程序中包含读取数据的语句
MODIFIES SQL DATA	表示子程序中包含写数据的语句
SQL SECURITY DEFINER	表示只有定义者才有权执行存储过程
SQL SECURITY INVOKER	表示调用者有权执行存储过程

需要注意的是，这些特征在创建、修改函数和存储过程时都可以具体地进行设置，这里就以修改 shop 数据库下的 proc 存储过程为例，将存储过程的执行者改为调用者，并设置注释信息，具体 SQL 语句及执行结果如下。

```
mysql> ALTER PROCEDURE proc
    -> SQL SECURITY INVOKER
    -> COMMENT '从商品分类表中获取大于指定 id 值的数据';
Query OK, 0 rows affected (0.00 sec)
```

修改完成后，查看 proc 存储过程的状态，具体 SQL 语句及执行结果如下。

```
mysql> SHOW PROCEDURE STATUS LIKE 'proc' \G
*************************** 1. row ***************************
                  Db: shop
                Name: proc
                Type: PROCEDURE
             Definer: root@localhost
            Modified: 2018-06-12 11:40:38
             Created: 2018-06-12 11:39:58
       Security_type: INVOKER
             Comment: 从商品分类表中获取大于指定 id 值的数据
character_set_client: gbk
collation_connection: gbk_chinese_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

从上述执行结果可以看出，查询出的 Security_type 字段和 Comment 字段保存的信息

已从默认值更改为修改后的数据。

另外，当数据库中存在废弃的存储过程时，需要使用 MySQL 提供的 DROP 语句删除存储过程，其基本语法格式如下。

```
DROP PROCEDURE [IF EXISTS] 存储过程名称
```

例如，删除 shop 数据库的存储过程 proc，具体 SQL 语句及执行结果如下。

```
mysql> DROP PROCEDURE IF EXISTS proc;
Query OK, 0 rows affected (0.00 sec)
```

10.2.4 存储过程的错误处理

在存储过程执行期间，可以对某些特定的错误代码、警告或异常进行定义，然后再针对这些错误添加处理程序进行处理。例如，退出当前程序块或继续执行。接下来将针对存储过程中的错误处理进行详细讲解。

1. 自定义错误名称

在编写存储过程时，可以使用 DECLARE 语句为指定的错误声明一个名称，其基本语法格式如下。

```
DECLARE 错误名称 CONDITION FOR [错误类型]
```

在上述语法中，错误类型有两种可选值，分别为 mysql_error_code 和 SQLSTATE [VALUE] sqlstate_value。前者是数值类型表示的错误代码，如 1148；后者是 5 个字符长度的错误代码，如 SQLSTATE '42000'。

为了读者更好地理解，下面为 ERROR1148(42000)服务器错误代码声明一个名称，具体 SQL 语句及执行结果如下。

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE proc()
    -> BEGIN
    ->    DECLARE command_not_allowed CONDITION FOR SQLSTATE '42000';
    -> END
    -> $$
Query OK, 0 rows affected (0.00 sec)
mysql> DELIMITER ;
```

上述语句中，DECLARE…CONDITION FOR 为需要处理的错误代码 42000 命名为 command_not_allowed，这个名称用于定义此错误的处理程序语句中。

另外，以上示例还可以使用 mysql_error_code 自定义错误名称，具体 SQL 语句如下。

```
DECLARE command_not_allowed CONDITION FOR 1148;
```

2. 错误的处理程序

为错误代码命名后，接下来需要使用 MySQL 提供的 DECLARE…HANDLER 语句为

其设置处理程序，其基本语法格式如下。

```
DECLARE 错误处理方式 HANDLER
FOR 错误类型 [, 错误类型] …
程序语句段
```

在上述语法中，MySQL 支持的错误处理方式有两种，一种是 CONTINUE(遇到错误不处理，继续执行)，另一种是 EXIT(遇到错误时马上退出)；程序语句段表示遇到定义的错误时，需要执行的存储过程代码段。FOR 后的错误类型可选值有 6 种，其中两种与 DECLARE…CONDITION FOR 语句的错误类型相同，另外 4 种类型如下所示。

(1) 使用 DECLARE…CONDITION FOR 语句声明的错误代码名称。

(2) SQLWARNING：表示所有以 01 开头的 SQLSTATE 错误代码。

(3) NOT FOUND：表示所有以 02 开头的 SQLSTATE 错误代码。

(4) SQLEXCEPTION：表示除以 01 或 02 开头外的所有 SQLSTATE 错误代码。

为了读者更好地理解，下面以示例的方式演示存储过程的错误处理，具体 SQL 语句及执行结果如下。

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE proc_demo()
    -> BEGIN
    ->     DECLARE CONTINUE HANDLER FOR SQLSTATE '23000'
    ->     SET @num = 1;
    ->     INSERT INTO sh_goods_category (id, name) VALUES (20, '运动');
    ->     SET @num = 2;
    ->     INSERT INTO sh_goods_category (id, name) VALUES (20, '运动');
    ->     SET @num = 3;
    -> END
    -> $$
Query OK, 0 rows affected (0.00 sec)
mysql> DELIMITER ;
```

在上述语句中，错误处理的语句要定义在 BEGIN…END 中，在程序代码之前。其中，SQLSTATE 错误代码 23000 表示表中含有重复的键时不能插入数据。SET 语句用于设置会话变量，这里读者了解即可，具体的内容会在下一节中详细讲解。

下面调用存储过程，并查询当前会话变量 num 的值，具体 SQL 语句及执行结果如下。

```
mysql> CALL proc_demo();
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @num;
+------+
| @num |
+------+
|   3  |
+------+
1 row in set (0.00 sec)
```

从上述变量的输出结果可知，当执行存储过程向表中插入重复的主键时，利用语句

DECLARE…HANDLER 跳过了此错误，程序继续执行，因此最后变量 num 的值为 3。

10.3 变量

MySQL 可以像编程语言一样使用，它提供了变量来保存数据。MySQL 中根据变量的作用范围可以将其划分为系统变量、会话变量和局部变量。本节将针对各种变量的操作进行详细讲解。

10.3.1 系统变量

系统变量也可称为全局变量，指的就是 MySQL 系统内部定义的变量，对所有 MySQL 客户端都有效。默认情况下，会在服务器启动时使用命令行上的选项或配置文件完成系统变量的设置。下面看一下如何在 MySQL 中查看并修改系统变量的值。

1. 查看系统所有变量

MySQL 提供了专门的语句查看系统所有变量，其基本语法格式如下。

```
SHOW [GLOBAL | SESSION] VARIABLES
[LIKE '匹配模式' | WHERE 表达式]
```

在上述语法中，修饰符 GLOBAL 用于显示全局系统变量值，当变量没有全局值时，则不显示任何值；SESSION 是默认的修饰符，可以使用 LOCAL 替换，也可以省略，用于显示当前连接中有效的系统可变值，如果变量没有会话值，则显示全局变量值。另外，SHOW VARIABLES 不带任何条件时则可以获取当前连接中系统所有有效的变量。

为了读者更好地理解，下面以查看变量名以 auto_inc 开头的系统变量值进行演示，具体 SQL 语句及执行结果如下。

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 1     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set, 1 warning (0.00 sec)
```

在上述执行结果中，查询到了 auto_increment_increment 和 auto_increment_offset 变量的值都为 1。自动增长字段 AUTO_INCREMENT 的值与这两个变量相关，具体计算方式请参考 MySQL 官方文档。

除此之外，在 MySQL 中还可以使用 SELECT 查看指定名称的当前系统变量值，具体 SQL 语句及执行结果如下。

```
mysql> SELECT @@query_cache_limit;
+---------------------+
| @@query_cache_limit |
```

```
+----------------------+
|       1048576        |
+----------------------+
1 row in set, 1 warning (0.00 sec)
```

上述 SQL 语句中，带有“@@”的变量，MySQL 会将其判断为系统变量或全局变量。

2. 修改系统变量的值

系统变量在成功连接 MySQL 服务器被初始化后，也可以根据实际需求对其进行修改，通常有局部修改和全局修改两种方式，具体如下所示。

（1）局部修改。若修改的系统变量只需在本次连接中有效，并且不影响其他连接 MySQL 服务器客户端的使用时，只能使用局部修改，基本语法格式如下。

```
SET 变量名 =新值;
```

例如，打开两个客户端，在一个客户端中使用 SET 将 auto_increment_offset 系统变量值设置为 5，然后在另一个客户端中查看该值是否有变化。具体 SQL 语句及执行结果如下。

```
#① 在客户端 1 中修改 auto_increment_offset 的值
mysql> SET auto_increment_offset =5;
Query OK, 0 rows affected (0.00 sec)
#② 在客户端 2 中查看 auto_increment_offset 的值
mysql> SHOW VARIABLES LIKE 'auto_increment_offset';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| auto_increment_offset | 1     |
+-----------------------+-------+
1 row in set, 1 warning (0.00 sec)
```

从上述的操作可知，系统变量的局部修改仅对执行修改操作的客户端连接有效，并不影响其他客户端。

（2）全局修改。全局方式修改系统变量，对所有正在连接的客户端无效，它只针对新连接的客户端永久生效。基本语法格式如下。

```
#语法 1
SET GLOBAL 变量名 =值;
#语法 2
SET @@GLOBAL.变量名 =值;
```

接下来，使用语法 1 对系统变量 auto_increment_offset 进行全局修改，然后再打开一个新的客户端连接 MySQL 服务器，查看此变量的变化，具体 SQL 语句及执行结果如下。

```
#① 在客户端 1 中全局修改 auto_increment_offset
mysql> SET GLOBAL auto_increment_offset =5;
Query OK, 0 rows affected (0.00 sec)
```

```
#② 在客户端 1 中查看 auto_increment_offset 的值
mysql> SHOW VARIABLES LIKE 'auto_increment_offset';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| auto_increment_offset | 1     |
+-----------------------+-------+
1 row in set, 1 warning (0.00 sec)
#③ 重新打开一个新的客户端 2,查看 auto_increment_offset 的值
mysql> SHOW VARIABLES LIKE 'auto_increment_offset';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| auto_increment_offset | 5     |
+-----------------------+-------+
1 row in set, 1 warning (0.00 sec)
```

从以上的操作中可以看出,全局修改系统变量后,对所有正在连接 MySQL 服务器的客户端来说,并没有变化,但会影响新打开的客户端。

10.3.2 会话变量

会话变量也可称为用户变量,指的是用户自定义的变量,跟 MySQL 当前客户端是绑定的,仅对当前用户使用的客户端生效。

会话变量是由"@"符号和变量名组成的,在定义会话变量时必须为该变量赋值,MySQL 中会话变量的赋值方式有 3 种,一是利用 SET 语句,二是在 SELECT 语句中利用赋值符号":="完成赋值,最后一种是利用 SELECT…INTO 语句。

为了让读者更好地理解,下面以示例的方式演示会话变量的定义与赋值。具体 SQL 语句及执行结果如下。

```
#方式 1:使用 SET 赋值
mysql> SET @name = 'Tom';
Query OK, 0 rows affected (0.00 sec)
#方式 2:在 SELECT 中使用赋值符号":="
mysql> SELECT @price :=price FROM sh_goods LIMIT 2;
+-----------------+
| @price :=price  |
+-----------------+
|           0.50  |
|          15.00  |
+-----------------+
2 rows in set (0.00 sec)
#方式 3:使用 SELECT…INTO 语句
mysql> SELECT id, name, price FROM sh_goods LIMIT 1
    -> INTO @ids, @names, @prices;
Query OK, 1 row affected (0.00 sec)
```

在上述 SQL 语句中,方式 1 利用 SET 和赋值运算符直接为定义的变量@name 赋值;

方式 2 展示了 sh_goods 中的 price 字段值为会话变量@price 赋值的过程，最后@price 的值为 15.00；方式 3 利用 SELECT…INTO 语句将查询的数据保存到变量@ids、@names、@prices 中。因为变量中只能保存一个数据，因此方式 3 查询的结果必须是一行记录，且记录中的字段个数必须与变量个数相同，否则系统会报错。

由于 MySQL 中的比较运算符与赋值运算符“＝”相同，为了避免系统分不清楚“＝”是比较运算符还是赋值运算符，强烈推荐为变量赋值时使用 MySQL 专门提供的赋值运算符“:＝”。

在设置完会话变量后，可以通过 SELECT 直接查询会话变量的值，具体 SQL 语句及执行结果如下。

```
mysql>SELECT @name, @price, @ids, @names, @prices;
+-------+--------+------+--------+---------+
| @name | @price | @ids | @names | @prices |
+-------+--------+------+--------+---------+
| Tom   | 15.00  |   1  |2B 铅笔  |   0.50  |
+-------+--------+------+--------+---------+
1 row in set (0.00 sec)
```

从上述执行结果可知，MySQL 中的变量只能保存一个数据。但当需要保存一组数据时，必须将其转换为 JSON 数据类型才可以保存一组数据。具体 SQL 语句及执行结果如下。

```
mysql>SELECT JSON_ARRAY(id,name), JSON_OBJECT(id,name)
    ->FROM sh_goods LIMIT 1
    ->INTO @arrinfo, @objinfo;
Query OK, 1 row affected (0.00 sec)
mysql>SELECT @arrinfo, @objinfo;
+--------------+----------------+
| @arrinfo     | @objinfo       |
+--------------+----------------+
|[1, "2B 铅笔"]  |{"1": "2B 铅笔"}  |
+--------------+----------------+
1 row in set (0.00 sec)
```

10.3.3　局部变量

相对于 MySQL 提供的系统变量和用户自定义的会话变量，局部变量的作用范围仅在复合语句语法 BEGIN 和 END 语句之间，保证局部变量在除 BEGIN 和 END 之间以外的任何地方，不能被获取和修改，方便在 MySQL 的函数和存储过程中保存需要操作的数据。

局部变量使用 DECLARE 语句定义，具体语法格式如下：

```
DECLARE 变量名 1 [, 变量名 2] … 数据类型 [DEFAULT 默认值]
```

上述语法中，局部变量的名称和数据类型是必选参数，当同时定义多个局部变量时，它们只能共用同一种数据类型。另外，DEFAULT 用于设置变量的默认值，省略时变量的初始值为 NULL。

接下来通过为函数创建局部变量为例，演示局部变量在程序中的使用，具体 SQL 语句及执行结果如下。

```
mysql>DELIMITER $$
mysql>CREATE FUNCTION func() RETURNS INT
    ->BEGIN
    ->   DECLARE age INT DEFAULT 10;
    ->   RETURN age;
    ->END
    ->$$
Query OK, 0 rows affected (0.01 sec)
mysql>DELIMITER ;
```

在上述 SQL 语句中，age 是局部变量的名称，int 是该变量的数据类型，10 是 age 的默认值。接下来调用 func()函数，具体 SQL 语句及执行结果如下。

```
mysql>SELECT func();
+---------+
| func()  |
+---------+
|    10   |
+---------+
1 row in set (0.00 sec)
```

从上述执行结果可知，通过返回函数结果可以在程序外查看到局部变量的值。但是在程序外直接使用 SELECT 则查询不到该变量，具体 SQL 语句及执行结果如下。

```
mysql>SELECT age;
ERROR 1054 (42S22): Unknown column 'age' in 'field list'
```

值得一提的是，局部变量的名称不区分大小写，它的命名规则与其他标识符相同，并且它的声明必须在游标或处理程序声明之前。

10.4 流程控制

在 MySQL 中除了可以自定义函数、存储过程、变量外，还可以使用流程控制根据特定的条件执行指定的 SQL 语句，或根据需要循环执行某些 SQL 语句。为此，MySQL 提供了多种流程控制语句，如 IF、CASE、LOOP、REPEAT、WHILE。接下来将对它们的具体使用和注意事项进行详细讲解。

10.4.1 判断语句

判断语句用于根据一些条件作出判断，从而决定执行指定的 SQL 语句。MySQL 中常用的判断语句有 IF 和 CASE 两种。

1. IF 语句

MySQL 中提供了两种不同的 IF 语法，一种适用于 SQL 语句中的条件判断，另外一种

用于函数、存储过程等程序中实现复杂的 SQL 操作。

（1）适用于 SQL 语句的 IF 语法。

```
IF(条件表达式, 表达式 1, 表达式 2)
```

在上述语法中，当条件表达式的值为 TRUE 时，则判断的结果返回表达式 1 的值，否则返回表达式 2 的值。需要注意的是，条件表达式不能是与 0 或 NULL 进行比较的表达式。

为了读者更好地理解，下面通过一个简单的案例演示 SELECT 语句中 IF 语法的使用，具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, name FROM sh_goods WHERE IF(score=5, score, 0);
+----+----------+
| id | name     |
+----+----------+
|  3 | 碳素笔   |
|  5 | 智能手机 |
+----+----------+
2 rows in set (0.01 sec)
```

上述 SQL 语句中，利用 IF 语句判断商品的 score 评分值是否等于 5，如果等于 5，则返回 score，WHERE 的判断条件为真，获取其对应的商品信息 id 和 name；当 score 评分值不等于 5 时，则返回 0，WHERE 的判断条件为假，不获取对应的商品信息 id 和 name。

（2）适用于 SQL 程序的 IF 语法。

```
IF 条件表达式 1 THEN 语句列表
[ELSEIF 条件表达式 2 THEN 语句列表] … [ELSE 语句列表]
END IF
```

在上述语法中，当条件表达式 1 为真时，执行对应 THEN 子句后的语句列表；条件表达式 1 为假时，继续判断条件表达式 2 是否为真，若为真，则执行其对应的 THEN 子句后的语句列表，依此类推。若所有条件表达式都为假，则执行 ELSE 子句后的语句列表。另外，每个语句列表必须由一个或多个 SQL 语句组成，且不允许为空。

为了读者更好地理解，下面通过一个简单的案例演示存储程序中 IF 语法的使用，具体 SQL 语句及执行结果如下。

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE isnull(IN val INT)
    -> BEGIN
    ->   IF val IS NULL
    ->     THEN SELECT 'THE parameter is NULL';
    ->   ELSE
    ->     SELECT 'THE parameter is not NULL';
    ->   END IF;
    -> END
    -> $$
Query OK, 0 rows affected (0.00 sec)
mysql> DELIMITER ;
```

在上述代码中，IF 用于判断为存储程序 isnull 的参数是否为空，当调用存储过程传递的参数为 NULL 时，则输出“THE parameter is NULL”，传递非 NULL 值(如 0)时，输出“THE parameter is not NULL”。

2. CASE 语句

MySQL 中提供了两种不同的 CASE 语法，一种适用于 SQL 语句中的条件判断，另外一种用于函数、存储过程等程序中实现复杂的 SQL 操作。

(1) 适用于 SQL 语句的 CASE 语法。

```
#语法 1
CASE 条件表达式 WHEN 表达式 1 THEN 结果 1
[WHEN 表达式 2 THEN 结果 2] … [ELSE 结果] END
#语法 2
CASE WHEN 条件表达式 1 THEN 结果 1
[WHEN 条件表达式 2 THEN 结果 2] … [ELSE 结果] END
```

在上述语法中，语法 1 和语法 2 的区别在于，前者使用 CASE 的条件表达式与 WHEN 后子句中的表达式进行比较，直到与其中的一个表达式相等时，则输出对应的 THEN 子句后的结果；而后者是直接判断 WHEN 后的条件表达式，直到其中一个的判断结果为真时，输出对应的 THEN 子句后的结果；若 WHEN 子句的表达式都不满足，则执行 ELSE 子句后的结果，另外当 CASE 语句中不含 ELSE 子句时，判断结果直接返回 NULL。

为了读者更好地理解，下面通过一个简单的案例演示 SELECT 语句中 CASE 语法的使用，具体 SQL 语句及执行结果如下。

```
mysql> SELECT id, name,
    -> (CASE WHEN price < 50 THEN '小额商品'
    -> WHEN price < 100 THEN '低价商品'
    -> WHEN price < 200 THEN '平价商品'
    -> ELSE '大额商品' END) AS desc_price
    -> FROM sh_goods;
+----+------------+------------+
| id | name       | desc_price |
+----+------------+------------+
|  1 | 2B 铅笔    | 小额商品   |
|  2 | 钢笔       | 小额商品   |
|  3 | 碳素笔     | 小额商品   |
|  4 | 超薄笔记本 | 大额商品   |
|  5 | 智能手机   | 大额商品   |
|  6 | 桌面音箱   | 低价商品   |
|  7 | 头戴耳机   | 平价商品   |
|  8 | 办公电脑   | 大额商品   |
|  9 | 收腰风衣   | 大额商品   |
| 10 | 薄毛衣     | 小额商品   |
+----+------------+------------+
10 rows in set (0.00 sec)
```

上述 SQL 语句中，利用 CASE 语法判断商品的 price 小于 50 的显示为“小额商品”，大

于等于 50 且小于 100 的显示为“低价商品”，大于等于 100 且小于 200 的显示为“平价商品”，其余大于等于 200 的显示为“大额商品”。

（2）适用 SQL 程序的 CASE 语法。

```
#语法 1
CASE 条件表达式 WHEN 表达式 1 THEN 语句列表
[WHEN 表达式 2 THEN 语句列表] … [ELSE 语句列表]
END CASE
#语法 2
CASE WHEN 条件表达式 1 THEN 语句列表
[WHEN 条件表达式 2 THEN 语句列表] … [ELSE 语句列表]
END CASE
```

以上的语法与适用于 SQL 语句的 CASE 语法不同的地方有以下几点。

（1）THEN 子句后执行的内容：前者的语句列表必须由一个或多个 SQL 语句组成，不可以为空；而后者的结果只能是一个表达式，不可以是 SQL 语句。

（2）CASE 的结束标识不同：前者使用 END CASE 结尾，而后者使用 END 结尾。

（3）当表达式的判断结果都为 FALSE，且 CASE 中没有设置 ELSE 子句时：前者执行时会返回“ERROR 1339 (20000): Case not found for CASE statement”错误；而后者执行时返回 NULL 值。

小提示：CASE 不能用于判断 NULL，这是由于 NULL 与 NULL 利用运算符“=”进行比较的结果为 FALSE。

为了读者更好地理解，下面通过一个简单的案例演示存储程序中 CASE 语法的使用，具体 SQL 语句及执行结果如下。

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE proc_level (IN score DECIMAL(5, 2))
    -> BEGIN
    ->    CASE
    ->    WHEN score > 89 THEN SELECT '优秀';
    ->    WHEN score > 79 THEN SELECT '良好';
    ->    WHEN score > 69 THEN SELECT '中等';
    ->    WHEN score > 59 THEN SELECT '及格';
    ->    ELSE SELECT '不及格';
    ->    END CASE;
    -> END
    -> $$
Query OK, 0 rows affected (0.00 sec)
mysql> DELIMITER ;
```

在上述存储过程中，利用 CASE 语句完成指定分数的级别判断。当传递的 score 参数值大于等于 90 时，显示结果为“优秀”；传递的 score 参数值大于等于 80 小于 90 时，显示结果为“良好”；传递的 score 参数值大于等于 70 小于 80 时，显示结果为“中等”；传递的 score 参数值大于等于 60 小于 70 时，显示结果为“及格”，其余小于 60 的显示“不及格”。

10.4.2 循环语句

循环语句指的是符合指定条件的情况下，重复执行一段代码。例如，计算给定区间内数据的累加和。MySQL 提供的循环语句有 LOOP、REPEAT 和 WHILE 三种。

1. LOOP 语句

LOOP 语句通常用于实现一个简单的循环，其基本语法格式如下。

```
[标签:] LOOP
  语句列表
END LOOP [标签];
```

在上述语法中，LOOP 用于重复执行语句列表，在语句列表中需要给出结束循环的条件，否则会出现死循环。通常情况下，使用判断语句进行条件判断，使用“LEAVE 标签”语句退出循环。其中，标签的定义只要符合 MySQL 标识符的定义规则即可。

为了读者更好地理解，接下来以计算 1～9 之间数字的和为例演示 LOOP 语句的使用。具体 SQL 语句及执行结果如下。

```
mysql>DELIMITER $$
mysql>CREATE PROCEDURE proc_sum()
    ->BEGIN
    ->  DECLARE i, sum INT DEFAULT 0;
    ->  sign: LOOP
    ->    IF i >=10 THEN
    ->        SELECT i, sum;
    ->        LEAVE sign;
    ->    ELSE
    ->      SET sum = sum +i;
    ->      SET i =i +1;
    ->    END IF;
    ->  END LOOP sign;
    ->END
    ->$$
Query OK, 0 rows affected (0.00 sec)
mysql>DELIMITER;
```

在上述程序中，局部变量 i 和 sum 的初始值都为 0，然后在 LOOP 语句中判断 i 的值是否大于等于 10，若是则输出当前 i 和 sum 的值，并退出循环；若不是则将 i 的值累加到 sum 变量中，并对 i 进行加 1，再次执行 LOOP 中的语句。

测试以上创建的存储程序，查看 LOOP 循环后 i 和 sum 的值，具体 SQL 语句及执行结果如下。

```
mysql>CALL proc_sum();
+------+------+
| i    | sum  |
+------+------+
```

```
|   10    |   45    |
+-------+-------+
1 row in set (0.00 sec)
Query OK, 0 rows affected (0.01 sec)
```

从上述执行结果可知，当 i 等于 10 时，不再对 sum 进行累加。因此，最后 sum 中保存的是 1～9 之间数字的累加和 45。

2. REPEAT 语句

MySQL 提供的 REPEAT 语句用于循环执行符合条件表达式的语句列表，其基本语法格式如下。

```
[标签:] REPEAT
  语句列表
UNTIL 条件表达式 END REPEAT [标签]
```

在上述语法中，程序会无条件执行一次 REPEAT 的语句列表，然后再判断 UNTIL 后的条件表达式是否为真，若为真，则结束循环；否则继续循环 REPEAT 的语句列表。

为了读者更好地理解，接下来以计算 10 以内的奇数和为例演示 REPEAT 语句的使用。具体 SQL 语句及执行结果如下。

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE proc_odd()
     -> BEGIN
     ->   DECLARE i, sum INT DEFAULT 0;
     ->   REPEAT
     ->     IF i % 2 != 0 THEN
     ->       SET sum = sum + i;
     ->     END IF;
     ->     SET i = i + 1;
     ->   UNTIL i > 10 END REPEAT;
     ->   SELECT i, sum;
     -> END
     -> $$
Query OK, 0 rows affected (0.00 sec)
mysql> DELIMITER;
```

在上述程序中，局部变量 i 和 sum 的初始值都为 0，利用 REPEAT 循环语句，当 i 为奇数时，将其累加到 sum 中，否则不进行累加。然后修改 i 的值，并判断 i 是否大于 10，当其大于 10 时，退出循环遍历。

下面调用存储过程并查看 i 和 sum 的值，具体 SQL 语句及执行结果如下。

```
mysql> CALL proc_odd();
+-------+-------+
| i     | sum   |
+-------+-------+
|   11  |   25  |
```

```
+------+------+
1 row in set (0.00 sec)
Query OK, 0 rows affected (0.01 sec)
```

从上述执行结果可知，局部变量 i 在 REPEAT 循环结束后变为 11，1～10 之间的奇数累加和为 25。

3. WHILE 语句

WHILE 语句用于创建一个带条件判断的循环过程，与 REPEAT 不同的是，WHILE 在语句执行时，需要满足条件表达式的要求，才会执行对应的语句列表，基本语法格式如下。

```
[标签:] WHILE 条件表达式 DO
  语句列表
END WHILE [标签]
```

在上述语法中，只要 WHILE 的条件表达式为真，就会重复执行 DO 后的语句列表。因此，如无特殊需求，一定要在 WHILE 的语句列表中设置循环出口，避免出现死循环。

为了读者更好地理解，下面以计算 1～10 之间的偶数和为例演示 WHILE 语句的使用。具体 SQL 语句及执行结果如下。

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE proc_even()
    -> BEGIN
    ->   DECLARE i, sum INT DEFAULT 0;
    ->   WHILE i<=10 DO
    ->     IF i % 2 = 0
    ->       THEN SET sum = sum + i;
    ->     END IF;
    ->     SET i=i+1;
    ->   END WHILE;
    ->   SELECT i, sum;
    -> END
    -> $$
Query OK, 0 rows affected (0.00 sec)
mysql> DELIMITER ;
```

在上述程序中，局部变量 i 初始值小于 10，因此执行 DO 后的语句列表，当 i 是偶数时将其累加到 sum 局部变量中，否则不进行累加。然后改变 i 的值，再次判断其是否小于 10，若还满足则继续重复以上的步骤，否则退出循环。

下面调用存储过程，查看 1～10 之间的偶数和，具体 SQL 语句及执行结果如下。

```
mysql> CALL proc_even();
+------+------+
| i    | sum  |
+------+------+
|  11  |  30  |
```

```
+-------+-------+
1 row in set (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
```

从上述执行结果可知,局部变量 i 在 WHILE 循环结束后变为 11,1～10 之间的偶数累加和为 30。

10.4.3　跳转语句

跳转语句用于实现程序执行过程中的流程跳转。MySQL 中常用的跳转语句有 LEAVE 和 ITERATE 语句。其基本语法格式如下。

```
{ITERATE | LEAVE} 标签名;
```

在上述语法中,ITERATE 语句用于结束本次循环的执行,开始下一轮循环的执行操作,重新开始循环;而 LEAVE 语句用于终止当前循环,跳出循环体。

为了读者更好地理解 LEAVE 和 ITERATE 语句的区别及使用,下面通过以下的示例进行对比演示。具体 SQL 语句及执行结果如下。

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE proc_jump()
    -> BEGIN
    ->   DECLARE num INT DEFAULT 0;
    ->   my_loop: LOOP
    ->     SET num = num + 2;
    ->     IF num < 5 THEN ITERATE my_loop;
    ->     ELSE SELECT num; LEAVE my_loop;
    ->     END IF;
    ->   END LOOP my_loop;
    -> END
    -> $$
Query OK, 0 rows affected (0.00 sec)
mysql> DELIMITER;
```

在上述程序中,局部变量 num 的初始值为 0,在 LOOP 循环中,当 num 小于 5 时,利用 ITERATE 不执行以下操作,重新开始 LOOP 循环,直到 num 大于 5 时,查看 num 的具体值,并利用 LEAVE 跳出 LOOP 循环。

接下来调用存储过程,查看跳出循环时 num 的值,具体 SQL 语句及执行结果如下。

```
mysql> CALL mydb.proc_jump();
+-------+
| num   |
+-------+
|   6   |
+-------+
1 row in set (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
```

需要注意的是，ITERATE 只能应用在循环结构 LOOP、REPEAT 和 WHILE 语句中，LEAVE 除可以在循环结构中应用外，还可在 BEGIN…END 中使用。

10.5 游标

10.5.1 游标的作用

在数据库的管理过程中，通过前面学习的 SELECT 语句仅能返回符合指定条件的结果集，但是没有办法对结果集中的数据进行下一行的检索或每次一条记录一条记录地单独处理等。此时，就可以利用 MySQL 提供的游标机制进行处理。

游标的本质是一种能从 SELECT 结果集中每次提取一条记录的指针，它主要用于交互式的应用程序，用户可以根据需要浏览或修改结果集中的数据。例如，现实生活中，在手机中查找与家人的聊天记录，需要使用“手”一条一条逐行翻看以找到想要查找的内容。这个过程与 MySQL 提供的游标机制类似，聊天记录可以看作是 SELECT 的查询结果集，“手”可以看作是数据库中的游标。

10.5.2 游标的操作流程

在 MySQL 中游标应用于函数和存储过程中，在使用时需要符合它的操作流程，一般将其分为 4 个步骤，定义游标、打开游标、利用游标检索数据和关闭游标。下面将对这 4 个操作步骤进行详细讲解。

1. 定义游标

游标在使用之前，必须通过定义，让其与指定的 SELECT 语句相关联，目的就是确定游标要操作的 SELECT 结果集对象。其基本语法格式如下。

```
DECLARE 游标名称 CURSOR FOR SELECT 语句
```

在上述语法中，游标名称必须唯一，因为在存储过程或函数中可以存在多个游标，而游标名称是唯一用于区分不同游标的标识。游标在定义时必须在错误处理程序的语句之前，局部变量声明之后。另外，SELECT 语句中不能含有 INTO 关键字。

需要注意的是，DECLARE…CURSOR FOR 语句定义游标后，与游标相关联的 SELECT 语句并没有执行，此时 MySQL 服务器的内存中并没有 SELECT 语句的查询结果集。

2. 打开游标

游标定义完成后，要想使用游标，首先需要打开游标，使 SELECT 语句根据查询条件将数据存储到 MySQL 服务器的内存中。其基本语法格式如下。

```
OPEN 游标名称
```

3. 利用游标检索数据

在打开游标之后，就可以利用 MySQL 提供的 FETCH 语句检索 SELECT 结果集中的

数据。每访问一次FETCH语句就获取一行记录，获取数据后游标的内部指针就会向前移动指向下一条记录，保证了每次获取的数据都不同。其基本语法格式如下。

```
FETCH [[NEXT] FROM] 游标名称 INTO 变量名 [, 变量名] …
```

在上述语法中，FETCH语句根据指定的游标名称将检索出来的数据存放到对应的变量中。另外，变量名的个数必须与声明游标时通过SELECT语句查询的结果集的字段个数保持一致。

需要注意的是，要想利用FETCH语句检索所有数据，需要利用循环语句实现，其中最常与游标一起使用的是REPEAT语句。当利用FETCH语句从游标中检索出最后一条记录后，再次执行FETCH语句，将产生“ERROR 1329(02000): No data to FETCH”错误信息。因此，在使用游标时通常利用DECLARE…HANDLER处理该错误，从而结束游标的循环遍历。

4. 关闭游标

游标检索完数据后，应该利用MySQL提供的语法关闭游标，释放游标占用的MySQL服务器的内存资源。其基本语法格式如下。

```
CLOSE 游标名称
```

在上述语法中，利用CLOSE关闭游标后，若再次需要利用游标检索数据时，仅需使用OPEN打开游标即可，不需要再次利用DECLARE…CURSOR FOR语句重新定义游标。值得一提的是，如果没有利用CLOSE关闭游标，它也会在到达程序最后的END语句的地方自动关闭。

10.5.3 使用游标检索数据

在了解了什么是游标、游标的作用以及操作流程后，接下来通过一个完整的案例演示游标检索数据的使用。假设在活动日，将库存不足400的5星评分商品的库存量增加到1500，利用shop数据库下的sh_goods表进行具体的操作，SQL语句及执行结果如下。

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE sh_goods_proc_cursor()
    -> BEGIN
    ->   DECLARE mark, cur_id, cur_num INT DEFAULT 0;
    ->   #定义游标
    ->   DECLARE cur CURSOR FOR
    ->   SELECT id, stock FROM sh_goods WHERE score=5;
    ->   #自定义错误处理程序,结束游标的遍历
    ->   DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET mark=1;
    ->   #打开游标
    ->   OPEN cur;
    ->   #遍历游标
    ->   REPEAT
    ->     #利用游标获取一行记录
```

```
    ->        FETCH cur INTO cur_id, cur_num;
    ->        #处理游标检索的数据
    ->        IF cur_num >=0 && cur_num <=400 THEN
    ->          SET cur_num=1500;
    ->          UPDATE sh_goods SET stock=cur_num WHERE id=cur_id;
    ->        END IF;
    ->     UNTIL mark END REPEAT;
    ->     #关闭游标
    ->     CLOSE cur;
    ->END
    ->$$
Query OK, 0 rows affected (0.00 sec)
mysql>DELIMITER;
```

上述程序中，游标 cur 与 sh_goods 表中 5 星评分商品的信息相关联。然后在打开游标后，通过 REPEAT 遍历游标，FETCH 取出游标的一行记录，并将其存入到局部变量 cur_id 和 cur_num 中，接着判断 5 星商品的库存是否不足 400，若符合要求，将 cur_num 的值修改为 1500，然后更新 sh_goods 表中对应的字段。直到所有数据都获取完成后，再次循环执行 FETCH 会发生代码为 02000 的错误，利用 DECLARE…HANDLER FOR 处理，并将局部变量 mark 设置为 1，结束 REPEAT 循环，最后关闭游标。

下面在调用存储过程之前，首先查看 shop 数据库下 sh_goods 中 5 星评分商品的库存信息。具体 SQL 语句及执行结果如下。

```
mysql>SELECT id, stock FROM sh_goods WHERE score=5;
+----+-------+
| id  | stock  |
+----+-------+
|  3  |   500  |
|  5  |     0  |
+----+-------+
2 rows in set (0.00 sec)
```

从以上的执行结果可知，id 号为 5 的 stock 库存不足 400。下面调用存储过程，修改以上信息的库存量。

```
mysql>CALL sh_goods_proc_cursor();
Query OK, 0 rows affected (0.01 sec)
mysql>SELECT id, stock FROM sh_goods WHERE score=5;
+----+-------+
| id  | stock  |
+----+-------+
|  3  |   500  |
|  5  |  1500  |
+----+-------+
2 rows in set (0.00 sec)
```

从上述的操作可以看出，sh_goods 表中 5 星评分商品的库存已由原来的 0 变为 1500。

10.6　触发器

10.6.1　触发器的概述

触发器可以看作是一种特殊类型的存储过程，它与存储过程的区别在于存储过程使用时需要调用，而触发器是在预先定义好的事件(如 INSERT、DELETE 等操作)发生时，才会被 MySQL 自动调用。

创建触发器时需要与数据表相关联，当表发生特定事件(如 INSERT、DELETE 等操作)时，就会自动执行触发器中提前预订好的 SQL 代码，实现插入数据前，强制检验或转换数据等操作，或是在触发器中代码执行错误后，撤销已执行成功的操作，保证数据的安全。因此，不难看出触发器在使用时的优点和缺点，具体内容如下。

(1) 优点。

① 触发器可以通过数据库中的相关表实现级联无痕更改操作。

② 保证数据安全，进行安全校验。

(2) 缺点。

① 触发器的使用会影响数据库的结构，同时增加了维护的复杂程度。

② 触发器的无痕操作会造成数据在程序(如 PHP、Java 等)层面不可控。

10.6.2　触发器的基本操作

触发器的基本操作包括创建触发器、查看触发器、触发器的触发和删除触发器。接下来对触发器的基本操作进行详细讲解。

1. 创建触发器

在创建触发器时需要指定触发器的操作对象——数据表，且该数据表不能是临时表或视图。基本语法格式如下。

```
CREATE TRIGGER 触发器名字 触发时机 触发事件 ON 表 FOR EACH ROW 触发顺序
BEGIN
  操作的内容
END
```

在上述语法中，指定数据库下的触发器名称必须唯一，也就是不同数据库下可以含有名称相同的触发器。“ON 表 FOR EACH ROW”指定触发器的操作对象。

触发时机表示数据表在发生变化前后的两种状态，触发事件表示激活触发器的操作类型，触发顺序表示指定同一个表中多个触发器的执行顺序，默认情况下按创建顺序激活，具体如表 10-12 所示。

表 10-12　创建触发器的选项

选　　项	可　选　值	描　　述
触发时机	BEFORE	在表中数据发生改变前的状态
	AFTER	在表中数据已经发生改变后的状态

续表

选　　项	可　选　值	描　　述
触发事件	INSERT	插入操作
	UPDATE	更新操作
	DELETE	删除操作
触发顺序	FOLLOWS 其他触发器名称	新触发器在现有触发器之后激活
	PRECEDES 其他触发器名称	新触发器在现有触发器之前激活

在表 10-12 中，对于每张数据表来说，每个触发事件只允许创建一个触发器。因此，一张数据表根据触发时机的不同最多支持 6 个触发器。

为了读者更好地理解，接下来以 shop 数据库中 sh_goods 商品表和 sh_user_shopcart 购物车表为例，简单演示用户商品添加购物车后自动减少对应商品的库存。其中，sh_user_shopcart 购物车中的 goods_id 字段表示商品 id，goods_num 字段表示某商品对应的购买件数。

为 sh_user_shopcart 购物车表创建触发器的 SQL 语句及执行结果如下。

```
mysql> DELIMITER $$
mysql> CREATE TRIGGER insert_tri BEFORE INSERT
    -> ON sh_user_shopcart FOR EACH ROW
    -> BEGIN
    ->   DECLARE stocks INT DEFAULT 0;
    ->   SELECT stock INTO stocks FROM sh_goods WHERE id=new.goods_id;
    ->   IF stocks <=new.goods_num THEN
    ->     SET new.goods_num :=stocks;
    ->     UPDATE sh_goods SET stock=0 WHERE id=new.goods_id;
    ->   ELSE
    ->     UPDATE sh_goods SET stock=stocks-new.goods_num WHERE id=new.goods_id;
    ->   END IF;
    -> END;
    -> $$
Query OK, 0 rows affected (0.01 sec)
mysql> DELIMITER;
```

上述语句为 sh_user_shopcar 表创建了一个触发器，在插入数据前首先判断购物车中的商品数量是否大于等于 sh_goods 商品表中商品的库存，若是则将待插入 sh_user_shopcart 表中的购买数量修改为此商品的最大库存，同时将 sh_goods 表中商品的库存修改为 0，若不是则直接修改 sh_goods 表中商品的库存。

值得一提的是，创建触发器时，可以使用 new 关键字获取插入或更新时产生的新值，使用 old 关键字获取删除或更新以前的值，使用方式很简单，只需在关键字后添加一个点“.”，然后再跟上具体的字段名称即可，如以上创建触发器时的 new. goods_id 和 new. goods_num，表示为 sh_user_shopcart 表插入数据前，获取要插入的 goods_id 值与 goods_num 值。

另外需要注意的是，old 关键字获取的字段值全部为只读形式，不能更新。

2. 查看触发器

若要查看指定数据库中已存在的触发器语句、状态等信息可以通过两种方式实现。一

种就是利用 MySQL 提供的 SHOW TRIGGERS 语句，另外一种就是利用 SELECT 直接查看 information_schema 数据库下的 triggers 数据表中的数据。这里以第 1 种方式进行讲解和演示，其基本语法格式如下。

```
SHOW TRIGGERS [{FROM | IN} 数据库名称] [LIKE '匹配模式' | WHERE 条件表达式]
```

在上述语法中，当未通过 FROM 或 IN 指定数据库时，SHOW TRIGGERS 获取的是当前选择的数据库下所有的触发器，WHERE 用于指定查看触发器的条件。另外，LIKE 子句的使用比较特殊，它用于匹配触发器作用的数据表，而非触发器名称。

为了读者更好地理解，接下来以查看 shop 数据库下的触发器为例进行演示。具体 SQL 语句及执行结果如下。

```
mysql> SHOW TRIGGERS\G
*************************** 1. row ***************************
             Trigger: insert_tri
               Event: INSERT
               Table: sh_user_shopcart
           Statement: BEGIN
DECLARE stocks INT DEFAULT 0;
SELECT stock INTO stocks FROM sh_goods WHERE id=new.goods_id;
IF stocks <=new.goods_num THEN
     SET new.goods_num :=stocks;
     UPDATE sh_goods SET stock=0 WHERE id=new.goods_id;
ELSE
UPDATE sh_goods SET stock=stocks-new.goods_num WHERE id=new.goods_id;
END IF;END
              Timing: BEFORE
             Created: 2018-08-03 11:04:39.03
            sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,
NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
             Definer: root@localhost
character_set_client: gbk
collation_connection: gbk_chinese_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.01 sec)
```

在上述执行结果中，Trigger 指定触发器的名称，Event 指定激活触发器的事件，Table 表示定义触发器的数据表，Statement 表示触发器激活时执行的语句，Timing 指定触发器在触发事件之前或之后激活。除此之外，还有创建触发器的日期时间 Created、触发器执行时有效的 SQL 模式以及创建触发器的账户信息等。

3. 触发器的触发

触发器在创建完成后，若使触发器触发，则需要让触发器指定的数据表执行设置的对应操作。为了让读者更好地理解，接下来以触发 insert_tri 触发器为例进行演示，具体 SQL 语句及执行结果如下。

（1）首先查看 sh_goods 表中商品编号为 5 的库存量 stock。

```
mysql> SELECT id, stock FROM sh_goods WHERE id=5;
+----+-------+
| id | stock |
+----+-------+
| 5  | 1500  |
+----+-------+
1 row in set (0.00 sec)
```

（2）接着向购物车表 sh_user_shopcart 中插入数据，然后触发设置的触发器。

```
mysql> INSERT INTO sh_user_shopcart
    -> (user_id, goods_id, goods_num, goods_price)
    -> VALUES (3, 5, 2000, 1999.00);
Query OK, 1 row affected (0.00 sec)
```

（3）最后查看 sh_goods 和 sh_user_shopcart 表在触发器触发后商品信息的变化。

```
mysql> SELECT id, stock FROM sh_goods WHERE id=5;
+----+-------+
| id | stock |
+----+-------+
| 5  |   0   |
+----+-------+
1 row in set (0.00 sec)
mysql> SELECT id, user_id, goods_id, goods_num, goods_price
    -> FROM sh_user_shopcart;
+----+---------+----------+-----------+-------------+
| id | user_id | goods_id | goods_num | goods_price |
+----+---------+----------+-----------+-------------+
| 1  |       3 |        5 |      1500 |     1999.00 |
+----+---------+----------+-----------+-------------+
1 row in set (0.00 sec)
```

从上述操作可以看出，触发器的触发过程是一个无痕操作。向 sh_user_shopcart 购物车表中插入数据时，触发了设置的触发器 insert_tri，“隐式地”将 sh_goods 表中 id 等于 5 的商品 stock 值设置为 0，同时也将插入 sh_user_shopcart 中 goods_num 的值修改为 stock 修改前的值 1500。

另外，当对建立触发器的数据表进行批量操作时，每一条数据的变化都会触发预先定义好的触发器语句的执行，如一次向 sh_user_shopcart 表中插入 3 条记录，则 sh_goods 表中对应的 3 条记录也都会发生相应的变化。

4. 删除触发器

删除触发器的操作很简单，只需使用 MySQL 提供的 DROP TRIGGER 语句即可。基本语法格式如下。

```
DROP TRIGGER [IF EXISTS] [数据库.]触发器名称
```

在上述语法中，利用“数据库.触发器名称”的方式可以删除指定数据库下的触发器。当省略“数据库.”时则删除当前选择的数据库下的触发器，若没选择数据库则系统会报错。

例如，删除 shop 数据库下的 insert_tri 触发器，具体 SQL 语句及执行结果如下。

```
mysql> DROP TRIGGER IF EXISTS insert_tri;
Query OK, 0 rows affected (0.00 sec)
```

除此之外，当删除数据表时，也会同时删除该表上创建的触发器。

10.7　事件

10.7.1　事件的概述

MySQL 中的事件指的是在某个特定的时间根据计划让其自动完成指定的任务或每隔多长时间根据计划做一次指定的任务。其中，在现实开发中，间隔一段时间完成设定的任务使用最多，它类似于 UNIX 中的定时任务(crontab)或 Windows 中的计划任务。例如，每天定时更新 3 天前的文章信息等。

事件是由 MySQL 提供的特殊事件调度程序执行与管理的，它适用于每隔一段时间就有固定需求的操作任务(如创建表、删除数据等)。而 MySQL 中的触发器与事件的区别在于前者仅针对某个表产生的事件(INSERT、UPDATE、DELETE)执行特定的任务，而后者是根据时间的推移触发设定的任务，并且操作对象可以是多张数据表。

10.7.2　事件的基本操作

1. 开启事件调度器

MySQL 中的事件是由事件调度器 Event Scheduler(特殊的线程)执行和管理的，默认情况下处于关闭状态。因此，若想要创建的事件能够正常地执行需要开启事件调度器，常用的查看和设置事件调度器的方式是利用全局变量 event_scheduler 实现的。具体 SQL 语句及执行结果如下。

```
#查看事件调度器是否开启
mysql> SHOW VARIABLES LIKE 'event_scheduler';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| event_scheduler | OFF   |
+-----------------+-------+
1 row in set, 1 warning (0.01 sec)
#设置事件调度器的状态
mysql> SET GLOBAL event_scheduler = ON;
Query OK, 0 rows affected (0.00 sec)
```

在上述 SQL 语句中，查看全局变量 event_scheduler 的值为 OFF(可以使用数字 0 代替)表示关闭；将其设置为 ON(可以使用数字 1 代替)，表示开启事件调度器。

除此之外，该变量还有一个值——DISABLED，用于表示不允许操作事件计划程序。但是 DISABLED 没有对应的数字表示，因此推荐读者在设置事件调度器时采用 ON 和 OFF。

值得一提的是，全局变量 event_scheduler 的操作方式还可以通过另外一种方式实现，读者可参考 10.3.1 节，这里不再赘述。

2. 创建事件

虽然 MySQL 中的事件信息都保存到 mysql.event 表中，但是不建议直接对该表进行操作，避免出现不可预知的错误。推荐直接使用 MySQL 提供的 CREATE EVENT 语句在指定的数据库下实现即可。其基本语法格式如下。

```
CREATE EVENT [IF NOT EXISTS] 事件名称
ON SCHEDULE 时间与频率
[ON COMPLETION [NOT] PRESERVE]
[ENABLE | DISABLE]
[COMMENT '事件的注释']
DO 事件执行的任务主体
```

在上述语法中，事件名称不区分大小写，且必须是一个最大长度不超过 64 个字符的有效 MySQL 标识符。“ON SCHEDULE 时间与频率”定义事件开始与结束的时间、执行的频率以及持续时间，具体内容会在下面详细讲解。

ON COMPLETION 用于定义事件一旦过期是否被立即删除，默认值为“NOT PRESERVE”，表示删除，设置为 PRESERVE 表示不删除。而 ENABLE 和 DISABLE 则可以指定当前创建的事件是否可用，默认为 ENABLE 表示可用，DISABLE 表示禁用。

COMMENT 用于为事件设置注释，方便阅读与维护。DO 后设置事件发生时执行的 SQL 语句，当 SQL 语句为多条时，需要放在 BEGIN…END 中。

事件在创建时根据时间和频率的不同设置，可以实现仅执行一次或定期重复操作，具体使用方式如下。

(1) 执行一次。CREATE EVENT 语句后的时间与频率设置成以下的形式，表示该事件仅执行一次。其基本语法格式如下。

```
AT 时间戳 [+ INTERVAL  时间间隔 时间单位] …
```

在上述语法中，时间戳必须包括日期和时间，时间间隔可以是任意的数字，可选的时间单位如表 10-13 所示。另外，读者可根据实际情况通过“+ INTERVAL 时间间隔 时间单位”任意组合时间间隔。

表 10-13 时间单位

YEAR	QUARTER	MONTH	DAY	HOUR
MINUTE	WEEK	SECOND	YEAR_MONTH	DAY_HOUR
DAY_MINUTE	DAY_SECOND	HOUR_MINUTE	HOUR_SECOND	MINUTE_SECOND

为了读者更好地理解，下面演示从现在开始 1 分钟 20 秒后向 sh_goods_category 中添加一条记录，具体 SQL 语句及执行结果如下。

```
mysql> CREATE EVENT insert_data_event
    -> ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 MINUTE
    -> + INTERVAL 20 SECOND
    -> DO INSERT INTO sh_goods_category(id, name) VALUES (50, '食品');
Query OK, 0 rows affected (0.00 sec)
```

在上述语句中，CURRENT_TIMESTAMP 表示获取当前的时间戳，然后在其后添加了两个时间间隔，让该事件在 1 分 20 秒后自动执行 DO 后的语句。

需要注意的是，时间与频率的设置不能为过期时间，否则 MySQL 会报 WARNING 提示错误。

(2) 定期重复操作。事件中时间与频率的设置最常使用的就是执行定期的重复操作。基本语法格式如下。

```
EVERY 时间间隔 时间单位
[STARTS 时间戳 [+ INTERVAL 时间间隔 时间单位] …]
[ENDS 时间戳 [+ INTERVAL 时间间隔 时间单位] …]
```

在上述语法中，EVERY 用于指定事件的执行频率，STARTS 指定事件何时开始重复，ENDS 指定事件何时结束重复。

为了读者更好地理解，下面演示从现在开始的 1 年时间内每天删除 sh_goods 表中 is_on_sale 为 0 且 update_time 大于 30 天的商品。具体 SQL 语句及执行结果如下。

(1) 创建事件。

```
mysql> CREATE EVENT IF NOT EXISTS delete_event
    -> ON SCHEDULE EVERY 1 DAY
    -> ENDS CURRENT_TIMESTAMP + INTERVAL 1 YEAR
    -> ON COMPLETION PRESERVE
    -> DO CALL delete_proc();
Query OK, 0 rows affected (0.00 sec)
```

在上述语句中，ON SCHEDULE EVERY 指定事件的发生频率为 1 天，ENDS 指定了从现在开始一年后结束此事件的重复执行，ON COMPLETION PRESERVE 设置当事件结束后，保存该事件。DO 后的语句表示事件发生时，调用自定义存储过程 delete_proc。

(2) 创建存储过程。

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE delete_proc()
    -> BEGIN
    ->     DELETE FROM sh_goods
    ->     WHERE TO_DAYS(NOW()) - TO_DAYS(update_time) >= 30 AND is_on_sale=0;
    -> END
    -> $$
Query OK, 0 rows affected (0.00 sec)
mysql> DELIMITER;
```

在以上程序中，TO_DAYS()用于将时间戳转换为天数，当 30 天未更新商品且 is_on_sale 的值为 0 时，删除此商品。

3. 查看事件

事件创建完成后，若要查看该事件的相关信息，可以使用 SHOW EVENTS 语句获取，具体 SQL 语句及执行结果如下。

```
mysql> SHOW EVENTS \G
*************************** 1. row ***************************
                  Db: shop
                Name: delete_event
             Definer: root@localhost
           Time zone: SYSTEM
                Type: RECURRING
          Execute at: NULL
      Interval value: 1
      Interval field: DAY
              Starts: 2018-07-03 16:33:04
                Ends: 2019-07-03 16:33:04
              Status: ENABLED
          Originator: 0
character_set_client: gbk
collation_connection: gbk_chinese_ci
  Database Collation: latin1_swedish_ci
2 row in set (0.00 sec)
```

在上述执行结果中，Type 字段用于保存当前事件是否是重复执行，值 RECURRING 表示重复，它的另外一个值为 ONE TIME，表示仅执行一次。

除此之外，读者还可以通过“SHOW CREATE EVENT 事件名”的方式查看事件的创建语句信息，它与查看表的创建语句使用类似，这里不再演示。

4. 修改事件

事件在创建后，可以利用 MySQL 提供的 ALTER EVENT 语句对其进行重命名、时间与频率的修改等操作。其基本语法格式如下。

```
ALTER EVENT event_name
[ON SCHEDULE 时间与频率]
[ON COMPLETION [NOT] PRESERVE]
[RENAME TO 新事件名称]
[ENABLE | DISABLE]
[COMMENT '事件的注释']
[DO 事件执行的任务主体]
```

从以上的语法可知，ALTER 语句除了比 CREATE 语句多一个“RENAME TO”重新为事件命名外，其他选项全部相同。

为了读者更好地理解，下面将 delete_event 事件名称和时间频率分别修改为 d_event 和从现在开始仅执行一次。具体 SQL 语句及执行结果如下。

```
mysql>ALTER EVENT delete_event
    ->ON SCHEDULE AT CURRENT_TIMESTAMP
    ->ON COMPLETION PRESERVE
    ->RENAME TO d_event
    ->DO CALL delete_proc();
Query OK, 0 rows affected (0.00 sec)
```

修改完成后，通过 SHOW EVENT 可以看出事件名称以及时间频率类型的变化。另外，修改事件操作最常应用于临时关闭事件。如下所示。

```
mysql>ALTER EVENT d_event DISABLE;
Query OK, 0 rows affected (0.00 sec)
```

修改后可以查看该事件，会发现此事件的状态信息显示为 DISABLE，暂停预先定义好的任务。

5. 删除事件

事件执行完成后，若不再需要继续使用，可以利用 MySQL 提供的 DROP 语句将其删除，基本语法格式如下。

```
DROP EVENT [IF EXISTS] 事件名称
```

上述语法中，当删除的事件正在执行时，也会立即停止执行，并从服务器中完成删除。例如，删除以上定义的 d_event 事件，具体 SQL 语句及执行结果如下。

```
mysql>DROP EVENT d_event;
Query OK, 0 rows affected (0.00 sec)
```

10.8　预处理 SQL 语句

随着技术的发展，MySQL 5.7 对服务器端提供了预处理 SQL 语句的功能。所谓预处理指的就是将 SQL 语句中的关键字(如 SELECT…FROM…)与数据(如字段名、数据表名)分离，使得针对不同数据的相同 SQL 语句的执行开销更小，同时又可防止 SQL 注入的攻击。

相比传统方式的 SQL 语句，在执行时每条 SQL 都需要经过分析、编译和优化的步骤。预处理方式则是利用客户端与服务器的二进制协议，预先编译一次客户端发送的 SQL 语句模板，然后再根据客户端发送给服务器相应数量的变量进行执行操作，并且针对一条 SQL 语句模板可以执行多次，还无须考虑数据中含有未转义的 SQL 引号和分隔符字符。

在 MySQL 中预处理语句的实现有 3 步，第 1 步是准备预处理的 SQL 模板，第 2 步是执行预处理的语句，第 3 步是在使用完预处理语句后对其进行释放。下面将分别对其进行详细讲解。

1. 准备预处理语句

对于预处理的 SQL 语句来说也是有一定限制的，原因在于不是所有的 SQL 都可以被

编译。例如，查看警告和错误的语句以及与警告和错误相关的系统变量等。常用的增、删、改、查的语句以及大部分的 SHOW 语句都是可以的，此处因篇幅有限，读者可参考手册查看预处理都支持哪些 SQL 语句。

预处理语句的基本语法格式如下所示。

```
PREPARE 预处理语句名称 FROM 预处理的 SQL 语句
```

在上述语法中，预处理语句名称不区分大小写，用于标识预处理的 SQL 模板，在后面进行执行或释放预处理语句时使用；预处理的 SQL 语句可以是一个字符串或包含 SQL 语句的会话变量，在此 SQL 模板中使用“?”作为需要动态改变的数据信息。但需要注意的是，“?”符号不能表示 SQL 关键字或标识符。

为了读者更好地理解，下面以动态查看 sh_goods 表中对应 id 的 name 和 price 为例进行演示。具体 SQL 语句及执行结果如下。

```
mysql> PREPARE stmt FROM 'SELECT name,price FROM sh_goods WHERE id=?';
Query OK, 0 rows affected (0.00 sec)
Statement prepared
```

在上述语句中，stmt 是 FROM 后的 SQL 语句名称，其中，“?”字符用于占位，此处可以根据客户端绑定的不同变量值获取不同的内容。从上述执行结果可知，当预处理语句执行成功后，MySQL 默认会给出“Statement prepared”的信息提示。

除此之外，以上预处理的 SQL 语句也可以写在一个会话变量中，具体如下。

```
SET @sql='SELECT name, price FROM sh_goods WHERE id=?';
PREPARE stmt FROM @sql;
```

小提示：

（1）当预处理语句的名称已存在时，若再次创建同名的预处理语句，则系统会先释放原来的语句，然后再重新创建。

（2）为防止同时创建太多的预处理语句，可以通过 max_prepared_stmt_count 系统变量限定预处理的最多数量。

（3）PREPARE 语句可以在存储过程中使用，但是不能在自定义函数或触发器中使用。

2. 执行预处理语句

在声明预处理语句后，要想其正常地运行，需要使用 EXECUTE 执行，其基本语法格式如下。

```
EXECUTE 预处理语句名称 [USING @变量名 [, @变量名] …]
```

上述语法中，在执行时若预处理语句中含有占位符“?”，则必须使用 USING 绑定对应数量的会话变量。

下面以执行名为 stmt 的预处理语句为例进行演示，具体 SQL 语句及执行结果如下。

```
mysql> SET @id=3;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql>EXECUTE stmt USING @id;
+--------+-------+
| name   | price |
+--------+-------+
| 碳素笔  | 1.00  |
+--------+-------+
1 row in set (0.00 sec)
```

从上可知，在执行预处理语句时，首先定义会话变量保存数据，然后在 SQL 模板中按从左到右的顺序将其绑定到符号“?”占用的位置，最后执行此 SQL 语句。

3. 释放预处理语句

预处理操作完成后，为了节约其占用的资源，建议使用以下的语句进行释放。其基本语法格式如下。

```
{DEALLOCATE | DROP} PREPARE 预处理语句名称
```

在上述语法中，使用 DEALLOCATE 或 DROP 语句都可以释放一个已声明的预处理语句，释放后再次执行预处理语句会导致错误的发生。

下面以释放名为 stmt 的预处理语句为例进行演示，具体 SQL 语句及执行结果如下。

```
mysql>DEALLOCATE PREPARE stmt;
Query OK, 0 rows affected (0.00 sec)
```

值得一提的是，预处理 SQL 语句属于会话级别的操作，因此它仅适用于创建预处理语句的当前用户会话，不适用于其他会话。同时在会话结束后，即使不执行 DEALLOCATE 或 DROP 语句，创建的预处理语句也会被自动释放。

10.9 动手实践：数据库编程实战

数据库的学习在于多看、多学、多想、多动手，只有将理论与实际相结合，才能够体现出数据管理与维护的重要性，展现知识学习的价值与力量。接下来请结合本章所学的知识完成数据库编程实战操作。

【实践目标】

此实践的目标就是能够根据文字提示，利用本章所学的知识，在存储过程中完成 sh_goods 表中数据分页查询功能。

【实践需求】

(1) 创建一个名为 page_proc 的存储过程，并为其设置传入的参数，参数分别为当前的页码数以及每页显示的最大记录数。

(2) 获取分页的总记录数，并利用传递的参数计算总页数，拼接分页的查询语句。

(3) 根据传递的参数查询指定分页的记录，如第 2 页的记录。

【动手实践】

1. 创建存储过程

根据【实践需求】(1)可知存储过程的名字为 page_proc，同时需要为此存储过程设置两个传入的参数。具体 SQL 语句如下。

```
DELIMITER $$
CREATE PROCEDURE page_proc(IN curp INT, IN per_page INT)
BEGIN
  #在此处完成数据编程
END;
$$
DELIMITER;
```

在上述语句中，参数 curp 表示当前的页码数，per_page 表示每页显示的最大记录数。然后在 BEGIN 和 END 之间完成相关的数据处理。

2. 获取分页的数据

将步骤 1 中的注释位置替换成以下的语句，完成数据的判断，以及指定分页数据的获取。具体 SQL 语句及执行结果如下。

```
#① 定义变量
DECLARE total_records, total_page, start INT;
#② 从指定的数据表中获取总记录数
SELECT COUNT(*) INTO total_records FROM sh_goods;
#③ 判断 per_page 是否符合要求
IF per_page<=1 THEN SET per_page=3;
END IF;
#④ 根据 total_records 和 per_page 计算总页数
SET total_page=CEIL(total_records / per_page);
#⑤ 判断 curp 是否符合要求
IF curp <1 THEN SET curp=1;
ELSEIF curp >total_page THEN SET curp=total_page;
END IF;
#⑥ 计算记录的偏移量
SET start=(curp -1) * per_page;
#⑦ 根据分页获取数据
SET @sql_stmt=CONCAT('SELECT * FROM sh_goods LIMIT', start,'\,' , per_page);
PREPARE paging FROM @sql_stmt;
EXECUTE paging;
DEALLOCATE PREPARE paging;
```

在上述语句中，首先从 shop 数据库的 sh_goods 表中获取总记录数，然后判断每页的记录数是否符合要求，若不符合将其设置为 3，最后计算总页数；接着判断当前页码是否符合要求，计算记录的偏移量，然后拼接执行的 SQL 语句，最后使用预处理方式完成 SQL 语句的执行。

3. 获取指定分页的记录

假设当前 sh_goods 表中一共有 10 条记录，完成存储过程创建后，可采用以下的方式调用该存储过程，查看是否可以获取指定页码的数据信息。

```
CALL page_proc(2,4);        #测试获取第 2 页的记录数,每页最多显示 4 条记录
CALL page_proc(-2,4);       #测试页码为负数的情况,每页最多显示 4 条记录
CALL page_proc(50,20);      #测试页码数超过实际页码数的情况,每页最多显示 20 条记录
CALL page_proc(3,1);        #测试获取第 3 页,每页展示最大记录数为小于等于 1 的情况
CALL page_proc(5,1);        #测试页码数过大,每页展示最大记录数为小于等于 1 的情况
CALL page_proc(-1,1);       #测试页码数为负,每页展示最大记录数为小于等于 1 的情况
```

由于篇幅有限，此处不再演示，读者可参考以上语句自己进行测试。

10.10　本章小结

本章主要讲解了 MySQL 内置函数的功能、使用方法及其注意事项；自定义函数与存储过程的作用与区别，变量的分类及作用域范围；触发器与定时事件的处理，预处理 SQL 语句的操作，还有如何在 MySQL 中对数据进行检索、遍历、判断等编程操作。通过本章的学习，读者应具备数据库基础编程的能力。

10.11　课后练习

一、填空题

1. 当前字符集为 GBK 时，函数 LENGTH('美丽的 jia')的结果为________。
2. 函数 SUBSTRING('bread',3)中的 3 表示截取的子串位置从________字母开始。
3. MySQL 用户定义的会话变量是由________和________组成的。
4. MySQL 中________循环语句会无条件执行一次语句列表。
5. MySQL 提供的________可自定义新的语句结束符号。

二、判断题

1. ITERATE 语句可以在 BEGIN…END 中实现语句跳转。(　　)
2. 对于所有用户来说，系统变量只能读取不能修改。(　　)
3. 只有 OPEN 打开游标后，查询结果才会存到 MySQL 服务器内存中。(　　)
4. MySQL 5.7 提供的预处理功能可以将 SQL 语句与数据分离。(　　)
5. 触发器的使用会影响数据库的结构，同时增加了维护的复杂程度。(　　)

三、选择题

1. 以下不能在 MySQL 中实现循环操作的语句是(　　)。

   A. CASE　　B. LOOP　　C. REPEAT　　D. WHILE

2. 下列选项中，触发器不能触发的事件是(　　)。

A. INSERT　　B. UPDATE　　C. DELETE　　D. SELECT

3. 以下选项(　　)可以在预处理语句中表示动态改变的数据。

A. *　　B. _　　C. :　　D. ?

4. 函数(　　)可以在字符串 book 中获取字母 o 第一次出现的位置。

A. INSERT()　　B. FIND_IN_SET()

C. INSTR()　　D. SUBSTRING()

5. 下面(　　)函数可以获取动态的实时时间。

A. NOW()　　B. SYSDATE()

C. CURRENT_TIMESTAMP()　　D. 以上答案全正确

四、简答题

1. 请简述存储过程和函数的区别。
2. 请简述触发器的基本语法以及其作用。

五、实训题

1. 请在 shop 数据库中创建一个存储过程，以订单编号为参数，输出该订单的商品信息。

2. 请在 shop.sh_order_goods 表上创建一个触发器，当添加订单-商品信息时，修改 sh_goods 表中对应商品的库存量。

第 11 章 数据库优化

学习目标

- 掌握 InnoDB 存储引擎的使用
- 掌握索引的操作及使用原则
- 熟悉表级锁与行级锁的区别
- 了解分区及分表技术的应用

对于数据库的操作应用，不仅要学会前面讲解的数据增删改查、视图、事务、存储过程的定义，还要能根据实际开发需求，合理安排资源，使 MySQL 运行更快。其中包括数据表创建时存储引擎的选择、索引的创建以及操作数据时锁机制的应用、分区分表技术的应用等。本章将针对 MySQL 的数据库优化进行详细讲解。

11.1 存储引擎

11.1.1 什么是存储引擎

简单地说，存储引擎可以看作是数据表存储数据的一种格式，不同的格式具有的特性也各不相同。例如，只有 InnoDB 存储引擎支持事务、外键、行级锁等特性，而 MyISAM 则支持压缩机制等特性。

存储引擎本身是 MySQL 数据库服务器的底层组件之一，最大的特点是采用“可插拔”的存储引擎架构。所谓“可插拔”指的是对正在运行的 MySQL 服务器依然可根据实际需求使用特定语句加载(插入，INSTALL PLUGIN 语句)或卸载(拔出，UNINSTALL PLUGIN 语句)所需的存储引擎文件。对于加载的文件则必须放在 MySQL 服务器插件目录中，若读者不清楚 MySQL 服务器插件目录，可通过查询 plugin_dir 系统变量获取。

可插拔存储引擎架构提供了一组标准的管理和支持服务，负责执行数据库的实际数据 I/O(输入/输出)操作，针对特定应用需求可以自由选择或自定义专用的存储引擎，而无须增加其他编码选项等额外的操作，从而达到可插拔存储引擎架构的目的——为特定的应用程序提供一组最优的选择，减少不必要的开销，增强数据库的性能。

11.1.2 存储引擎的选择

由于 MySQL 中提供了多种存储引擎，每种存储引擎的特性又各不相同，那么在创建数据表时选择一个合适的 MySQL 存储引擎，则会对整体系统效率和性能产生巨大影响。

在了解各存储引擎的特点之前，首先看一下 MySQL 当前支持哪些存储引擎。具体 SQL 语句如下。

```
SHOW ENGINES;
```

执行以上 SQL 语句，运行的结果中含有 6 个字段，分别为 Engine（存储引擎）、Support（是否支持）、Comment（注释说明）、Transactions（是否支持事务）、XA（是否支持分布式事务）和 Savepoints（是否支持事务的保存点设置）。

为方便读者阅读与学习，下面以表格的形式进行展示，如表 11-1 所示。

表 11-1　MySQL 默认支持的存储引擎

存储引擎	默认是否支持	是否支持事务	是否支持分布式事务	是否支持保存点	描　述
InnoDB	DEFAULT	YES	YES	YES	支持事务、行级锁和外键
MyISAM	YES	NO	NO	NO	支持表锁、全文索引
MRG_MYISAM	YES	NO	NO	NO	相同 MyISAM 表的集合
MEMORY	YES	NO	NO	NO	内存存储，速度快但数据容易丢失，适用于临时表
CSV	YES	NO	NO	NO	数据以文本方式存储在文件中
ARCHIVE	YES	NO	NO	NO	适用存储海量数据，有压缩功能，但不支持索引
BLACKHOLE	YES	NO	NO	NO	黑洞引擎，写入的数据都会消失，适合做中继存储
PERFORMANCE_SCHEMA	YES	NO	NO	NO	适用于性能架构
FEDERATED	NO	NULL	NULL	NULL	用于访问远程的 MySQL 数据库

为了读者更好地理解，下面分别对表 11-1 中的存储引擎进行介绍。

（1）InnoDB 存储引擎。InnoDB 存储引擎在 MySQL 5.7 版本中被指定为默认的存储引擎，用于完成事务、回滚、崩溃修复和多版本并发控制的事务安全处理。同时，也是 MySQL 中第一个提供外键约束的表引擎，尤其对事务处理的能力是 MySQL 其他存储引擎无法与之比拟的。

InnoDB 存储引擎的优势在于提供了良好的事务管理、崩溃修复能力和并发控制，缺点是其读写效率一般。

（2）MyISAM 存储引擎。MyISAM 存储引擎是基于 ISAM 存储引擎发展起来的，它不仅解决了 ISAM 的很多不足，还增加了很多有用的扩展。例如，数据的全文索引、压缩与加密、支持复制与备份的恢复等。在 MySQL 5.5 以前的版本中是 MySQL 的默认存储引擎，与 InnoDB 存储引擎相比，MyISAM 的优点是处理速度快，缺点是不支持事务的完整性和并发性。

（3）MRG_MYISAM 存储引擎。MRG_MYISAM 存储引擎也被称为 MERGE 存储引擎，指的是相同 MyISAM 存储引擎表的集合。这意味着，所有合并的表必须具有相同顺序

的字段与索引的应用。它具有可以快速拆分大型只读表,执行搜索效率更高等优势;同时也具有索引读取速度较慢、只能针对 MyISAM 存储引擎的表进行合并等劣势。因此,实际应用中此存储引擎的应用较少。

值得一提的是,MRG_MYISAM 存储引擎的功能与分区技术的功能基本相同,可以进行替换。关于分区的实现会在后面小节中详细讲解。

(4) MEMORY 存储引擎。MEMORY 是 MySQL 中一种特殊的存储引擎。在 MEMORY 存储引擎的表中,所有数据都保存在内存中,因此数据的处理速度快。

此存储引擎的特点是,一旦程序出错或服务器断电都会导致数据的丢失,所以不适合持久保存数据,而且也不能存储太大的数据。对于需要很快的读写速度,但数据量小、不需要持久保存的临时数据来说,MEMORY 存储引擎是一个理想的选择。

(5) CSV 存储引擎。CSV 是采用文本方式存储数据的一种存储引擎,数据在文件中通过逗号分隔保存。对于使用 CSV 存储引擎的数据表,会被保存成 3 个文件,文件名与表名相同,文件扩展名分别为 frm(存储表结构信息)、csv(存储表内容)和 csm(存储表的状态、数据量等元数据)。CSV 存储引擎在使用时还有两点需要注意,一是不支持索引和分区,二是 CSV 表中的所有字段必须含有 NOT NULL 属性。

(6) ARCHIVE 存储引擎。ARCHIVE 存储引擎适合保存数量庞大、长期维护但很少被访问的数据。对于使用 ARCHIVE 存储引擎的数据表,数据存储时会利用 zlib 压缩库进行压缩,在记录被请求时会实时进行解压。需要注意的是,ARCHIVE 存储引擎仅仅支持查询和插入操作,并且因不支持数据索引,查询效率较低。

(7) BLACKHOLE 存储引擎。BLACKHOLE 存储引擎也可称为"黑洞"存储引擎,它的特点是写入的数据都会消失,就像被"黑洞"吞噬了一样。利用此特性可以将其作为转发器或过滤器。例如,将一个主服务器中的大量数据,经过过滤后再搬到从服务器,可以将 BLACKHOLE 存储引擎的数据表作为过滤器使用,且不会保存任何数据,但是它会在二进制日志中记录下所有的 SQL 语句,然后可以复制并执行这些语句,将结果保存到从服务器中。

(8) PERFORMANCE_SCHEMA 存储引擎。MySQL 5.7 中有一个名为 performance_schema 的数据库,在该数据库下保存的所有数据表的存储引擎都为 PERFORMANCE_SCHEMA,主要用于收集数据库服务器性能参数。所以用户不能为数据表创建此类型的存储引擎,对于读者来说只需了解即可。

(9) FEDERATED 存储引擎。默认情况下,FEDERATED 存储引擎在 MySQL 中不可用。因此,在使用时需要利用"--federated"选项进行启动,才可以创建从远程 MySQL 服务器访问数据的表。其中,本地的 FEDERATED 表只保存结构信息(后缀为 frm),远程服务器同时要保存结构信息和数据文件,所有的增删改查操作都通过访问远程服务器后,才将结果返回给本地的服务器。

11.1.3 InnoDB 存储引擎

在 MySQL 5.7 中,若创建的数据表没有指定存储引擎,则默认使用 InnnoDB。它是具有高可靠性和高性能的通用存储引擎,具有提交、回滚和崩溃恢复功能的事务处理能力、提高多用户并发处理的能力以及维护数据完整性等优势,非常适合业务逻辑比较强,修改操作比较多的项目。例如,常见的电子商务网站、办公系统等。下面将从 InnoDB 表的存储格

式、表空间大小的设置以及多版本控制 3 个方面进行详细讲解。

1. 存储格式

默认情况下，使用 InnoDB 存储引擎创建的数据表都共用一个表空间文件 ibdata1，通常位于 data 目录下，与数据库文件处于同级目录。ibdata1 用于集中存储数据和索引；另外，每张数据表都会在对应的数据库下创建一个与表同名的结构文件（数据库/表名.frm）。

若想要修改默认设置，使每一个 InnoDB 表都有一个独立的表空间文件（数据库/表名.idb），则可通过全局变量 innodb_file_per_table 实现。具体 SQL 语句如下。

```
#① 查看默认是否共用同一个表空间文件
SHOW VARIABLES LIKE 'innodb_file_per_table';
#② 开启每个表独立的表空间文件
SET GLOBAL innodb_file_per_table=ON;
```

在上述语句中，若①查询的结果为 OFF，表示所有 InnoDB 表都共用同一个表空间文件；若想要每张数据表都有一个独立的表空间文件，可按照②的方式实现。需要注意的是，全局变量 innodb_file_per_table 值的变化，不会影响已经含有独立表空间的表。

2. 增加表空间大小

数据在实际的存储过程中，可能会遇到存储空间不足的情况，此时可采用 MySQL 提供的方式扩充 InnoDB 系统表空间的大小。对于 InnoDB 表来说，增加表空间的大小通常有两种方式，一种是将其配置为自动扩展，另外一种是在表空间达到指定大小后，将数据存储到另一个文件中。

（1）自动扩展表空间。自动扩展表空间的实现方式非常简单，只需为表空间中最后一个数据文件（利用系统变量 innodb_data_file_path 获取）指定 autoextend 属性，每次自动增加的空间大小由系统变量 innodb_autoextend_increment 设置，以兆字节（MB）为单位。具体 SQL 语句及执行结果如下。

```
mysql> SHOW VARIABLES LIKE 'innodb_data_file_path';
+-----------------------+------------------------+
| Variable_name         | Value                  |
+-----------------------+------------------------+
| innodb_data_file_path | ibdata1:12M:autoextend |
+-----------------------+------------------------+
1 row in set, 1 warning (0.00 sec)
mysql> SHOW VARIABLES LIKE 'innodb_autoextend_increment';
+-----------------------------+-------+
| Variable_name               | Value |
+-----------------------------+-------+
| innodb_autoextend_increment | 64    |
+-----------------------------+-------+
1 row in set, 1 warning (0.00 sec)
```

从上面的执行结果可知，InnoDB 表默认情况下采用自动扩展表空间的方式，每次扩充的大小为 64MB。该值也可根据实际情况进行自定义，如改为 10MB。

（2）存储到不同文件。在表空间中存储的数据达到上限时，也可以将数据存到另外一个指定的文件中。实现时先关闭 MySQL 服务器，然后去掉 innodb_data_file_path 系统变量的 autoextend 属性，将文件 ibdata1 可以存储的数据大小设置为一个固定值，接着在其后添加分号“;”以及另外一个文件的路径和大小，同时也可为新增的文件设置自动扩展功能，最后启动 MySQL 服务器即可。

为了读者更好地理解，下面演示 ibdata1 达到 12M 时，将数据添加到另一个指定的文件 ibdata2 中，当 ibdata2 达到 50M 时再自动扩展。具体操作如下。

第 1 步：关闭 MySQL 服务器，打开 MySQL 配置文件 my.ini，添加以下配置。

```
innodb_data_file_path=ibdata1:12M;ibdata2:50M:autoextend
```

第 2 步：开启 MySQL 服务器，在 ibdata1 同目录下可看到新创建的文件 ibdata2，然后登录 MySQL 服务器后，查看 innodb_data_file_path 系统变量已修改为以上设置的值。

3. 多版本控制

InnoDB 是一个多版本控制的存储引擎，它通过保存更改前的数据信息来处理多用户并发、事务回滚等情况的发生，从而保证数据读取的一致性。多版本控制的内部实现原理是 InnoDB 存储引擎会为每条记录添加 3 个隐藏的字段，分别为 DB_TRX_ID、DB_ROLL_PTR 和 DB_ROW_ID，具体含义如下。

- DB_TRX_ID 字段：表示最后一个插入或更新此记录的事务标识符，其中删除操作也被视为更新操作。
- DB_ROLL_PTR 字段：表示滚动指针，用于指向 MySQL 中撤销日志的记录，用于事务的回滚操作，并在事务提交后会立即删除。
- DB_ROW_ID 字段：用于保存新增记录的 ID。

以上就是 InnoDB 存储引擎能够实现多版本控制的基本原理，此处读者了解即可。

11.1.4　MyISAM 存储引擎

MyISAM 存储引擎在 MySQL 5.5 以前一直被设置为默认的存储引擎，随着技术的发展现已被功能更加强大的 InnoDB 所取代。尽管如此，MyISAM 相比 InnoDB 还有其独特的优势，比如表占用的空间小，数据写入速度快等特点。因此，它更适用于论坛、博客等写入读取操作非常多的系统。

对于使用 MyISAM 存储引擎的数据表，会被存储成 3 个文件，文件名与表名相同，文件扩展名分别为 frm、myd 和 myi，具体描述如表 11-2 所示。

表 11-2　MyISAM 存储引擎的相关文件

扩　展　名	功 能 说 明
frm	用于存储表的结构
myd	用于存储数据，是 MYData 的缩写
myi	用于存储索引，是 MYIndex 的缩写

值得一提的是，MyISAM 表的数据移植非常方便，只需将数据库下表 11-2 中对应的 3 个文件复制到另一个数据库下即可。

小提示：对于数据在表中的存储，MyISAM 与 InnoDB 存储引擎的存储格式不同。原因在于 MyISAM 表采用"堆组织"的方式存储数据，换言之，就是数据在 MyISAM 表中的保存顺序与插入顺序完全相同；而 InnoDB 表则采用"索引组织"方式存储数据，也就是数据会按照主键的顺序将记录显示到对应的位置，即使没有主键 InnoDB 也会自动选择表中符合条件的字段或采用 InnoDB 内置的 ROWID 作为主键。

11.2 索引

在现实生活中，为了方便快速地在书籍中找到待查找的内容，都会在书籍的开始添加一个目录，让用户可根据目录的内容与指定的页数快速定位到要查看的内容。同样地，为了快速在大量数据中找到指定的数据，可以使用 MySQL 提供的索引功能，让用户在执行查询操作时可以根据字段中建立的索引，快速地定位到具体位置。下面本节将对索引的分类、基本的操作以及使用原则进行详细讲解。

11.2.1 索引概述

索引是一种特殊的数据结构，可以看作是利用 MySQL 提供的语法将数据表中的某个或某些字段与记录的位置建立一个对应的关系，并按照一定的顺序排序好，类似于书籍中的目录，目的就是为了快速定位指定数据的位置。

根据索引实现语法的不同，MySQL 中常见的索引大致可以分为 5 种，具体描述如下。

- 普通索引：是 MySQL 中的基本索引类型，使用 KEY 或 INDEX 定义，不需要添加任何限制条件，作用是加快对数据的访问速度。
- 唯一性索引：由 UNIQUE INDEX 定义，创建唯一性索引的字段需要添加唯一性约束，用于防止用户添加重复的值。
- 主键索引：由 PRIMARY KEY 定义的一种特殊的唯一性索引，用于根据主键自身的唯一性标识每条记录，防止添加主键索引的字段值重复或为 NULL。另外，若在 InnoDB 表中数据保存的顺序与主键索引字段的顺序一致时，可将这种主键索引称为"聚簇索引"。一般聚簇索引指的都是表的主键，因此，一张数据表中只能有一个聚簇索引。
- 全文索引：由 FULLTEXT INDEX 定义，用于根据查询字符提高数据量较大的字段查询速度。此索引在定义时字段类型必须是 CHAR、VARCHAR 或 TEXT 中的一种，在 MySQL 5.7 版本中，仅 MyISAM 和 InnoDB 存储引擎支持全文索引。
- 空间索引：由 SPATIAL INDEX 定义在空间数据类型字段上的索引，提高系统获取空间数据的效率。其中，空间数据类型读者可参考手册。另外，在 MySQL 中仅有 MyISAM 和 InnoDB 存储引擎支持空间索引，还要保证创建索引的字段不能为空。

对于以上讲解的 5 种索引类型，根据创建索引的字段个数，还可以将它们分为单列索引和复合索引。单列索引指的是在表中单个字段上创建的索引，可以是普通索引、唯一索引、主键索引或者全文索引，只要保证该索引对应表中一个字段即可。而复合索引则是在表的多个字段上创建一个索引，且只有在查询条件中使用了这些字段中的第一个字段时，该索引

才会被使用。

值得一提的是，若创建的索引是从左开始截取数据表中字段值的一部分内容，这种索引可以统称为前缀索引。根据索引开始的部分字符，可以大大节约索引空间，提高索引的效率。形如在汉语词典中根据偏旁部首（类似前缀索引）查找汉字（要索引的数据）。

11.2.2　索引的基本操作

1. 创建索引

MySQL 中索引可以在创建数据表（CREATE TABLE）或对已创建的数据表进行添加（ALTER TABLE 或 CREATE INDEX）。其中，向已创建的数据表添加索引时，CREATE INDEX 语法不能向数据表添加主键索引，可以使用 ALTER TABLE 语法实现。其基本语法格式如下。

```
#方式 1:CREATE TABLE 创建数据表时添加索引
CREATE TABLE   数据表名(
    字段名 数据类型 [约束条件]
    …
    PRIMARY KEY [索引类型] (字段列表)[索引选项],
    {INDEX|KEY} [索引名称] [索引类型] (字段列表)[索引选项],
    UNIQUE [INDEX|KEY] [索引名称] [索引类型] (字段列表)[索引选项],
    {FULLTEXT|SPATIAL} [INDEX|KEY] [索引名称] (字段列表)[索引选项]
)[表选项];
#方式 2:ALTER TABLE 向已创建的数据表添加索引
ALTER TABLE 数据表名
    ADD PRIMARY KEY[索引类型] (字段列表)[索引选项]
    |ADD {INDEX|KEY} [索引名称] [索引类型] (字段列表)[索引选项]
    |ADD UNIQUE [INDEX|KEY] [索引名称] [索引类型] (字段列表)[索引选项]
    |ADD FULLTEXT [INDEX|KEY] [索引名称] (字段列表)[索引选项]
    |ADD SPATIAL [INDEX|KEY] [索引名称] (字段列表)[索引选项], …;
#方式 3:CREATE INDEX 向已创建的数据表添加索引
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX 索引名称
    [索引类型] ON 数据表名 (字段列表)[索引选项][算法选项 | 锁选项]
```

上述语法看似复杂，其实在使用时只需要确定 3 点即可，一是确定采用哪种方式创建索引，选择对应的语句；二是创建哪些索引，三是为各索引设置选项，具体如表 11-3 所示。其中，主键索引不能设置索引名称，其他索引的名称也可以省略，默认使用建立索引的字段表示，复合索引则使用第一个字段的名称作为索引名。

表 11-3　创建索引的选项设置

索 引 选 项	语　　法
索引类型	USING {BTREE | HASH}
字段列表	字段[(长度)][ASC | DESC]]
索引选项	KEY_BLOCK_SIZE [=]值 |索引类型| WITH PARSER 解析器插件名 | COMMENT '描述信息'

续表

索引选项	语　　法
算法选项	ALGORITHM [=] {DEFAULT|INPLACE|COPY}
锁选项	LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}

在上述索引选项中，只有字段列表是必选项，其余均为可选的。其中，索引选项中 KEY_BLOCK_SIZE 仅可在 MyISAM 存储引擎的表中使用，表示索引的大小（以字节为单位），WITH PARSER 只能用于全文索引。此外，对于全文和空间索引不能设置索引类型，而索引类型在不同的存储引擎中也不相同，如 InnoDB 和 MyISAM 支持 BTREE，而 MEMORY 则同时支持 BTREE 和 HASH。

为了读者更好地理解，下面以 ALTER TABLE 语法为例在 shop 数据库的 sh_goods 表中演示各种索引的创建。具体内容如下。

（1）普通索引与主键索引。

```
mysql>ALTER TABLE sh_goods ADD INDEX name_index (name);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

在上述语句中，INDEX 表示创建的索引为普通索引，name_index 表示为索引定义的名称，name 表示在 sh_goods 表的 name 字段上创建的索引。索引创建完成后，可以利用前面学习过的 SHOW CREATE TABLE 语句查看到指定表中创建的索引信息。具体 SQL 语句及执行结果如下。

```
mysql>SHOW CREATE TABLE sh_goods\G
*************************** 1. row ***************************
       Table: sh_goods
Create Table: CREATE TABLE `sh_goods` (
  ……此处省略字段的创建信息
  PRIMARY KEY (`id`),
  KEY `name_index` (`name`)
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

从上述执行结果可知，KEY 后的 name_index 就是上面创建的普通索引，而 PRIMARY KEY 是在创建 sh_goods 表时添加的主键约束，同时系统内部会自动为其创建主键索引，区别在于约束用于维护数据库完整性的规则，而索引则是为了方便对数据进行检索。除此之外，MySQL 还有唯一性约束和唯一性索引、外键约束（外键索引），读者在设置时也要明确两者的区别。

（2）唯一性索引。唯一性索引可在表中同时创建多个，但是创建唯一性索引的字段值不能重复，否则会创建不成功。具体 SQL 语句及执行结果如下。

```
mysql>ALTER TABLE sh_goods
    ->ADD UNIQUE INDEX unique_index (id);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

在上述语句中，UNIQUE INDEX 用于为 sh_goods 表的 id 字段设置唯一性索引。

(3) 全文索引。全文索引的功能与 SQL 中模糊查询类似，在实现时不支持前缀索引，因此在设置时可以省略字段长度的设定。具体 SQL 语句及执行结果如下。

```
mysql>ALTER TABLE sh_goods
    ->ADD FULLTEXT INDEX ft_index (content);
Query OK, 0 rows affected, 1 warning (0.12 sec)
Records: 0  Duplicates: 0  Warnings: 1
```

按照以上的语法创建完成后，在查询数据时若想要使用全文索引，需采用 MySQL 提供的特定语法，具体如下。

```
MATCH (字段列表) AGAINST (字符串)
```

在上述语法中，字段列表与创建全文索引的列表相同，并且 MySQL 默认情况下仅可以对英文和数字进行检索，大小写不敏感。字符串指的就是要查找的内容，该内容必须是字段值中一个完整的单词或句子，不能是单词中的一部分，否则会查不到指定的内容，这可以看作是全文索引与 SQL 中模糊查询不同的地方。

例如，content 字段的值为“very good”，则字符串只有为 very、good 或 very good 中的一个时才会用全文索引。

(4) 空间索引。在 MySQL 5.7 中，InnoDB 和 MyISAM 存储引擎都支持空间索引，但在创建时要保证索引字段不能为空。而 sh_goods 表中没有空间数据类型的字段，下面在 mydb 数据库下创建一个测试表 sp_table，用于演示空间索引的创建，具体 SQL 语句及执行结果如下。

```
#① 创建数据表
mysql>CREATE TABLE mydb.sp_table (space GEOMETRY NOT NULL);
Query OK, 0 rows affected (0.01 sec)
#② 添加空间索引
mysql>ALTER TABLE mydb.sp_table  ADD SPATIAL INDEX(space);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

在上述语句中，为 mydb. sp_table 表的 space 字段创建空间索引时，要保证该字段的数据类型必须为空间数据类型。其中，GEOMETRY 是具有层次结构的几何空间类型。

(5) 单列索引与复合索引。单列索引就是仅对一个字段设定的索引，在上面讲解的普通索引、主键索引、唯一性索引以及全文索引都是单例索引，这里不再演示。

与之相对的是复合索引，即同时在多个字段上创建一个索引，多个字段的设置顺序要遵循“最左前缀原则”，也就是把最频繁使用的字段放在最左侧，然后依此类推。具体 SQL 语句及执行结果如下。

```
mysql>ALTER TABLE sh_goods  ADD INDEX multi(name,price,keyword);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

在上述语句中，定义的复合索引是一个普通索引，multi 是复合索引的名称，而 name、price 和 keyword 是建立复合索引的字段。在查询时，只有第 1 个字段 name 被使用时，该复合索引才会被使用。

(6) 前缀索引。在实际应用中，有时需要设置索引的字段值是一个很长的字符(如 TEXT、很长的 VARCHAR 等)，这时会导致索引变大且检索速度也会变慢。解决办法是可以采用前缀索引的方式，在保证较高查询效率的同时避免空间的浪费。而在设置前缀索引时，字段长度的设定需要通过一定的计算与测试才能够选取最合适的一个范围。这里使用"不重复的索引数量/总记录数"的方法，比对查询时不设置长度与设置长度最相近的值，然后利用获取的这个值设置前缀索引。

为了读者更好地理解，下面依据 sh_goods 表的结构创建 mydb.temp 表，并保证 mydb.temp 表准备好足够的数据来进行测试。测试步骤如下。

```
#第 1 步:获取不设置长度时,不重复的索引数量/总记录数的值:0.1296
SELECT COUNT(DISTINCT keyword)/COUNT(keyword) FROM temp;
#第 2 步:设置长度,不重复的索引数量/总记录数的值:0.0926、0.1111、0.1296、0.1296
SELECT COUNT(DISTINCT LEFT(keyword, 1))/COUNT(keyword) FROM temp;
SELECT COUNT(DISTINCT LEFT(keyword, 2))/COUNT(keyword) FROM temp;
SELECT COUNT(DISTINCT LEFT(keyword, 3))/COUNT(keyword) FROM temp;
SELECT COUNT(DISTINCT LEFT(keyword, 4))/COUNT(keyword) FROM temp;
```

在上述语句中，在不设置长度时，获取的值为 0.1296；设置长度时，利用 LEFT()函数从指定字段中截取指定位数(如 1、2、3、4)的字符，然后计算获取的比值依次为 0.0926、0.1111、0.1296、0.1296。

因此，可以看出当截取 keyword 的前 3 位和 4 位字符时与不设置长度获取的比值相同(实际开发有可能不同，但获取的值会非常的近似)，同时也要尽可能地选取最小值。所以在 sh_goods 表中根据 keyword 字段设置的前缀索引长度就可以设置为 3。具体 SQL 语句及执行结果如下。

```
mysql>ALTER TABLE sh_goods ADD INDEX (keyword(3));
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

值得一提的是，在 MySQL 的索引分类中全文索引和空间索引不支持前缀索引的设置。

2. 查看索引

对于已经创建好的索引，除了以上使用的 SHOW CREATE TABLE 外，MySQL 还提供了其他几种方式查看与分析索引语句。其基本语法格式如下。

```
#语法 1
SHOW {INDEXES|INDEX|KEYS} FROM 表名;
#语法 2
{EXPLAIN | DESCRIBE | DESC}
{SELECT | DELETE | INSERT | REPLACE | UPDATE} statement
```

在上述语法中，语法 1 通常用于查看指定表中的索引信息，如索引名称、添加索引的字

段、索引类型等。语法 2 通常用于分析执行的 SQL 语句，且 SQL 语句仅可以是以上语法中出现的 5 种。

此外，虽然对于 MySQL 而言，EXPLAIN、DESCRIBE 和 DESC 表示的含义相同，但是在实际应用中，通常使用 DESCRIBE 和 DESC 获取表结构相关的信息，EXPLAIN 用于获取执行查询的相关数据，如是否引用索引，可能用到的索引等。

为了读者更好地理解，下面分别演示语法 1 和语法 2 的使用，具体如下。

（1）查询表中的所有索引信息。以查看 sh_goods 表中的所有索引信息为例，具体 SQL 语句及执行结果如下。

```
mysql> SHOW INDEX FROM sh_goods \G
*************************** 1. row ***************************
        Table: sh_goods
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
  Column_name: id
    Collation: A
  Cardinality: 10
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
此处省略了 7 行记录
8 rows in set (0.00 sec)
```

上述执行结果共有 8 行记录，每行记录的字段相同，这里以第一行记录为例进行讲解，其中主要字段的含义如表 11-4 所示。

表 11-4　索引信息的描述字段

字　段　名	描　　述
Non_unique	索引是否可以重复，0 表示不可以，1 表示可以
Key_name	索引的名字，如果索引是主键索引，则它的名字为 PRIMARY
Seq_in_index	建立索引的字段序号值，默认从 1 开始
Column_name	建立索引的字段
Collation	索引字段是否有排序，A 表示排序，NULL 表示没有排序
Cardinality	计算 MySQL 连接时使用索引的可能性（精确度不高），值越大，可能性越高
Sub_part	前缀索引的长度，如 3，若字段值都被索引则为 NULL
Index_type	索引类型，可选值有 BTREE、FULLTEXT、HASH、RTREE
Comment	索引字段的注释信息
Index_comment	创建索引时添加的注释信息

在表 11-4 中，当创建的索引是复合索引时，这些字段从左开始第 1 个的 Seq_in_index 值为 1，然后递增加 1 作为第 2 个字段的序号值，依次类推。Sub_part 字段只有在建立前缀索引时其值为设置的字段长度，否则为 NULL。

(2) 分析语句执行情况。MySQL 中 EXPLAIN 可以分析的语句有 SELECT、DELETE、INSERT、REPLACE 和 UPDATE。下面对查询 sh_goods 表中 name 以"笔"结尾的所有值为例进行分析，具体 SQL 语句及执行结果如下。

```
mysql>EXPLAIN SELECT name FROM sh_goods WHERE name='%笔' \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: sh_goods
   partitions: NULL
         type: ref
possible_keys: name_index,multi
          key: name_index
      key_len: 362
          ref: const
         rows: 1
     filtered: 100.00
        Extra: Using index
1 row in set, 1 warning (0.00 sec)
```

在上述执行结果中，possible_keys 表示此查询可能用到的索引，key 表示实际查询用到的索引。其余的相关字段描述如表 11-5 所示。

表 11-5　分析执行语句的字段

字　段　名	描　　述
id	查询标识符，默认从 1 开始，若使用了联合查询，则该值依次递增，联合查询结果对应的该值为 NULL
select_type	操作类型，如 DELETE、UPDATE 等，但是当执行 SELECT 语句时，它的值有多种，如 SIMPLE 表示不需联合查询或简单的子查询
table	输出数据的表
partitions	匹配的分区
type	连接的类型，如 const 使用了主键索引或唯一性索引，ref 表示使用前缀索引或条件中含有运算符"＝"或"＜＝＞"等
key_len	索引字段的长度
ref	表示哪些字段或常量与索引进行了比较，如 const 表示常量与索引进行了比较
rows	预计需要检索的记录数
filtered	按条件过滤的百分比
Extra	附加信息，如 Using index 表示使用了索引覆盖

在表 11-5 中，字段 select_type 和 type 的值还有很多，读者可自行查看手册进行了解。Extra 字段的值为 Using index 时，表示查询出现了索引覆盖。所谓索引覆盖指的是查询的

字段恰好是索引的一部分或与索引完全一致，那么查询只需要在索引区上进行，不需要到数据区检索数据的情况。这种查询的特点是速度非常快，但同时也会增加索引文件的大小，只有此索引的使用率尽可能高的情况下，索引覆盖才有意义。否则，在使用时应该避免此情况的发生。

3. 删除索引

对于数据表中已经创建但不再使用的索引，应该及时删除，避免占用系统资源，影响数据库的性能。MySQL 索引的删除分为两种情况，一种是删除主键索引，另外一种是删除非主键索引。

（1）删除主键索引。MySQL 中主键索引在删除时，需要考虑该主键字段是否含有 AUTO_INCREMENT 属性，若有则需在删除主键索引前删除该属性，否则程序会报以下的错误提示信息。

```
ERROR 1075 (42000): Incorrect table definition; there can be only one auto column and it must be defined as a key
```

删除含有 AUTO_INCREMENT 属性的主键索引，具体步骤和语法如下。

```
#① 删除主键字段的 AUTO_INCREMENT 属性
ALTER TABLE 数据表 MODIFY 字段名 字段类型
#② 删除主键索引
ALTER TABLE 数据表 DROP PRIMARY KEY 或 DROP INDEX `PRIMARY` ON 数据表
```

在上述语法中，当使用 DROP INDEX 删除主键索引时，其后的 PRIMARY 由于是 MySQL 中的保留字，因此必须使用反引号(`)包裹。

（2）删除非主键索引。在 MySQL 中删除主键索引以外的其他索引时，根据索引名称采用以下语法中的任意一种都可以完成删除操作。其基本语法格式如下。

```
#语法 1
ALTER TABLE 数据表 DROP INDEX 索引名
#语法 2
DROP INDEX 索引名 ON 数据表 [算法选项][锁选项]
```

在上述语法中，DROP 在删除索引时的算法选项与锁选项与创建索引时相同，请参照表 11-3。

例如，删除 sh_goods 表创建的普通索引 name_index。具体如下。

```
mysql> DROP INDEX name_index ON sh_goods;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

11.2.3　索引的使用原则

索引在使用时虽然可以提高查询速度，降低服务器的负载，但是相应的，索引的使用也会占用物理空间，给数据的维护造成很多麻烦，并且在创建和维护索引时，其消耗的时间会

随着数据量的增加而增长。因此，索引的使用还需遵从一些最基本的原则。

（1）查询条件中频繁使用的字段适合建立索引。

建立索引的目的就是为了快速定位指定数据的位置，所以在创建索引时，要选择会在WHERE子句、GROUP BY子句、ORDER BY子句或表与表之间连接时频繁使用的字段。例如，商品表中的价格字段经常用于筛选操作等，因此在实际开发时可酌情考虑是否给该字段添加索引，而提示字段则基本不会出现在查询语句中，因此一般不建议在这类字段上建立索引，避免消耗系统的空间。

（2）数字型的字段适合建立索引。

建立索引的字段类型也会影响查询和连接的性能。例如，数字型字段与字符串字段在处理时，前者仅需比较一次就可以了，而后者则需要逐个比较字符串中的每一个字符。因此，与数字型字段相比，字符串字段的执行时间更长，复杂程度也更高。在开发时一般建议尽可能地选择数字类型的字段建立索引。

（3）存储空间较小的字段适合建立索引。

MySQL中适用于存储数据的对应类型有多种选择，此时对于建立索引的字段来说，占用存储空间越少的越合适。例如，存储大量文本信息的TEXT类型与存储指定长度字符串的CHAR类型相比，显然CHAR类型，更有利于提高检索的效率。所以建立索引时推荐选择占用存储空间较小的字段。

（4）重复值较高的字段不适合建立索引。

在建立索引时，若字段中保存的数据重复值较高，在这种情况下，即使该字段（如性别字段）在查询时会被频繁使用，此时也不适合建立索引。以InnoDB为例，非主键索引在查询时，都需要先获取其对应的聚簇索引后才能完成数据的检索，因此当重复值较高时，需要重复获取相同聚簇索引检索数据的次数也会急剧增多，影响查询的效率。在开发时一般不推荐在重复值较高的字段建立索引。

（5）更新频繁的字段不适合建立索引。

对于建立索引的字段，数据在更新时，为了保证索引数据的准确性，同时还要更新索引。所以在字段被频繁更新时，会造成I/O访问量增加，影响系统的资源消耗，加重了存储的负载。

另外，对于已经创建索引的数据表来说，要想对该表的查询使用到索引，也有以下几点需要注意的，否则MySQL可能不会按预想的那样使用索引检索数据。

（1）查询时保证字段的独立。

对于建立索引的字段，在查询时要保证该字段在关系运算符（如＝、＞等）的一侧“独立”。所谓“独立”指的是索引字段不能是表达式的一部分或函数的参数。

例如，在sh_goods表中查询时，如WHERE条件表达式为id＋2＞3，那么即使id上已建立主键索引，查询时也不会使用，而若WHERE条件表达式为id＞3－2，那么查询就会使用主键索引，读者可以使用EXPLAIN语句进行分析，观察结果中key字段对应的值，这里不再演示。

（2）模糊查询中通配符的使用。

在模糊查询时，若匹配模式中的最左侧含有通配符（％），表示只要数据中含所有“％后指定的内容”就符合要求，所以会导致MySQL全表扫描，而不会使用设置的索引。例如，对

于 sh_goods 表中的索引，WHERE 子句中“name LIKE '笔记%'”就会使用复合索引，而“name LIKE '%笔记%'”就会放弃使用索引，采用全表扫描的方式查询。

值得一提的是，MySQL 的优化器在查询时会判断全表扫描是否会比使用索引慢，若是则使用索引，否则会使用全表扫描。

(3) 分组查询时排序的设置。

在 MySQL 中，分组查询默认情况下对分组的字段进行排序，所以在开发时若想要避免分组排序对性能的消耗，可以在分组后使用 ORDER BY NULL 禁止排序。

值得的一提的是，以上介绍的创建与使用索引的原则并不是一成不变的，读者需要结合开发经验设计出最符合开发需求的方式实现。

11.3 锁机制

11.3.1 认识锁机制

在认识锁机制前，首先思考一个问题：在同一时刻，用户 A 和用户 B 同时要获取并修改 sh_goods 表中 id 等于 2 的 stock 库存量值，此时会发生什么呢？

假设在初始情况下，sh_goods 表中 id 等于 2 的 stock 库存量值为 500。

在不添加锁的前提下，用户 A 关闭自动提交，将 stock 的值修改为 300，然后查询当前 stock 值为 300(修改但未提交)；与此同时用户 B 也获取 stock，它的值却为 500。当用户 A 提交了修改后，用户 B 获取到的值又变为 300。整个操作过程出现了两个大的问题，一是用户 B 第 1 次查询 stock 字段的值与用户 A 不同，二是用户 B 前后两次读取的 stock 值不同，从而产生了用户并发操作时数据不一致的情况。

解决办法就是，在用户 A 和用户 B 同时向 sh_goods 表发出请求操作时，根据系统内部设定的操作优先级(获取数据优先或修改数据优先的原则)，锁住指定用户(如 A)要操作的资源(sh_goods 表)，同时让另外一个用户(如 B)排队等候，直到锁定资源的用户(如 A)操作完成，并释放锁后，再让另一个用户(如 B)对资源进行操作。其中，对资源加锁的方式，可以采用修改事务隔离级别(前面章节已讲)的方式实现。

简单地说，锁机制就是为了保证多用户并发操作时，能使被操作的数据资源保持一致性的设计规则。又因 MySQL 数据库自身设计的特点，利用多种存储引擎处理不同特定的应用场景，所以锁机制在不同存储引擎中的表现也有一定的区别。

根据存储引擎的不同，MySQL 中常见的锁有两种，分别为表级锁(如 MyISAM、MEMORY 存储引擎)和行级锁(如 InnoDB 存储引擎)。另外 InnoDB 存储引擎中还含有表级锁，具体内容会在后面的小节详细讲解，此处了解即可。

表级锁是 MySQL 中锁的作用范围(锁的粒度)最大的一种锁，它锁定的是用户操作资源所在的整个数据表，有效地避免了死锁的发生，且具有加锁速度快、消耗资源小的特点。正所谓事物都有两面性，表级锁的优势同样给它带来了一定的缺陷，因其锁定的粒度大，在并发操作时发生锁冲突的概率也大。

行级锁是 MySQL 中锁的作用范围最小的一种锁，它仅锁定用户操作所涉及的记录资源，有效地减少了锁定资源竞争的发生，具有较高处理并发操作的能力，提升系统的整体性

能。但同时也因其锁定的粒度过小,每次加锁和解锁所消耗的资源也会更多,发生死锁的可能性更高。

另外,根据锁在 MySQL 中的状态也可将其分为"隐式"与"显式"。所谓"隐式"锁指的是 MySQL 服务器本身对数据资源的争用进行管理,它完全由服务器自动执行。而"显式"锁指的是用户根据实际需求,对操作的数据显式地添加锁,同样在使用完数据资源后也需要用户对其进行解锁。

小提示: 在了解死锁前,首先要理解什么是锁等待。所谓锁等待指的是一个用户(线程)等待其他用户(线程)释放锁的过程。而死锁可以简单地理解为两个或多个用户(线程)在互相等待对方释放锁而出现的一种"僵持"状态,若无外力作用,它们将永远处于锁等待的状态,此时就可以说系统产生了死锁或处于死锁状态。

11.3.2 表级锁

在实际应用中,表级锁根据操作的不同可以分为读锁和写锁。读锁表示用户读取(如 SELECT 查询)数据资源时添加的锁,此时其他用户虽然不可以修改或增加数据资源,但是可以读取该数据资源,因此读锁也可以称为共享锁;而写锁表示用户对数据资源执行写(如 INSERT、UPDATE、DELETE 等)操作时添加的锁,此时除了当前添加写锁的用户外,其他用户都不能对其进行读/写操作,因此写锁也可以称为排他锁或独占锁。

MyISAM 存储引擎表是 MySQL 数据库中最典型的表级锁,下面就以此存储引擎的表级锁为例详细讲解"隐式"读/写的表级锁和"显式"读/写表级锁的添加。

1. "隐式"读/写的表级锁

当用户对 MyISAM 存储引擎表执行 SELECT 查询操作前,服务器会"自动"地为其添加一个表级的读锁,执行 INSERT、UPDATE、DELETE 等写操作前,服务器会"自动"地为其添加一个表级的写锁;直到查询完毕,服务器再"自动"地为其解锁。执行时间可以看作是"隐式"表级锁读/写的生命周期,且该生命周期的持续时间一般都比较短暂。

默认情况下,服务器在"自动"添加"隐式"锁时,表的更新操作优先级高于表的查询操作。在添加写锁时,若表中没有任何锁则添加,否则将其插入到写锁等待的队列中;在添加读锁时,若表中没有写锁则添加,否则将其插入到读锁等待的队列中。

2. "显式"读/写的表级锁

在实际应用中,可以根据开发需求,对要操作的数据表进行"显式"地添加表级锁。其基本语法格式如下。

```
LOCK TABLES 数据表名 READ [LOCAL]| WRITE, …
```

在上述语法中,LOCK TABLES 可以同时锁定多张数据表。READ 表示表级的读锁,添加此锁的用户可以读取该表但不能对此表进行写操作,否则系统会报错;此时其他用户可以读取此表,若执行对此表的写操作则会进入等待队列。WRITE 表示表级的写锁,添加此锁的用户可以对该表进行读/写操作,在释放锁之前,不允许其他用户访问与操作。

需要注意的是，在为 MyISAM 存储引擎表设置“显式”读锁时，若添加 LOCAL 关键字，则在不发生锁冲突的情况下，未添加此锁的其他用户可以在表的末尾实现并发插入数据的功能。

此外，对于表级锁来说，虽然锁本身消耗的资源很少，但是锁定的粒度却很大，当多个用户访问时，会造成锁资源的竞争，降低了并发处理的能力。因此，从数据库优化的角度来考虑，应该尽量减少表级锁定的时间，进而提高多用户的并发能力。此时，对于用户添加的“显式”表级锁，需要使用 MySQL 提供的 UNLOCK TABLES 语句释放锁。

值得一提的是，用户设置的“显式”表级锁仅在当前会话内有效，若会话期间内未释放锁，在会话结束后也会自动释放。

为了读者更好地理解，下面通过一个具体的案例进行演示。具体步骤如下。

(1) 创建 MyISAM 表并插入 2 条测试数据。

```
mysql> CREATE TABLE mydb.table_lock(id int)ENGINE=MyISAM;
Query OK, 0 rows affected (0.01 sec)
mysql> INSERT INTO mydb.table_lock VALUES(1),(2);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

(2) 设置“显式”读的表级锁。

打开两个客户端 A 和 B，在客户端 A 中为 mydb.table_lock 设置“显式”读的表级锁后，然后分别在客户端 A 和客户端 B 中执行 SELECT 和 UPDATE 操作。具体 SQL 语句及执行结果如下。

```
#① 在客户端 A 中添加表级读锁
mysql> LOCK TABLE mydb.table_lock READ;
Query OK, 0 rows affected (0.00 sec)
#② 在客户端 A 中执行 SELECT 和 UPDATE 操作
mysql> SELECT * FROM mydb.table_lock \G
*************************** 1. row ***************************
id: 1
*************************** 2. row ***************************
id: 2
2 rows in set (0.00 sec)
mysql> UPDATE mydb.table_lock SET id=3 WHERE id=1;
ERROR 1099 (HY000): Table 'table_lock' was locked with a READ lock and can't
be updated
mysql> SELECT * FROM mydb.mt \G
ERROR 1100 (HY000): Table 'mt' was not locked with LOCK TABLES
#③ 在客户端 B 中执行 SELECT 和 UPDATE 操作
mysql> SELECT * FROM mydb.table_lock \G
*************************** 1. row ***************************
id: 1
*************************** 2. row ***************************
id: 2
2 rows in set (0.00 sec)
mysql> UPDATE mydb.table_lock SET id=3 WHERE id=1;
#此处光标会不停闪烁，进入锁等待状态
```

从以上的操作可以看出，添加表级读锁的客户端 A 仅能对 mydb. table_lock 执行读取操作，不能执行写操作，也不能操作其他未锁定的数据表，如 mydb. mt。对于未添加锁的客户端 B 则可以执行 SELECT 操作，但是执行 UPDATE 操作则会进入锁等待状态，只有客户端 A 结束会话或执行 UNLOCK TABLES 释放锁时，客户端 B 的操作才会被执行。具体 SQL 语句及执行结果如下。

```
#④ 在客户端 A 中释放锁
mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)
#⑤ 客户端 B 在客户端 A 释放锁后,会立即执行③中等待的写锁
mysql> UPDATE mydb.table_lock SET id=3 WHERE id=1;
Query OK, 1 row affected (5.64 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

（3）并发插入操作。

在 MyISAM 存储引擎的数据表中，还支持并发插入操作，用于减少读操作与写操作对表的竞争情况。实现语法为 LOCK…READ LOCAL，具体 SQL 语句及执行结果如下。

```
#① 在客户端 A 中添加表级读锁
mysql> LOCK TABLE mydb.table_lock READ LOCAL;
Query OK, 0 rows affected (0.00 sec)
#② 在客户端 B 中,插入一条记录
mysql> INSERT INTO mydb.table_lock VALUES(4);
Query OK, 1 row affected (0.00 sec)
```

从上述执行结果可知，即使客户端 A 中已添加了表级读锁，在未释放此读锁时，在客户端 B 中依然可以实现数据插入操作，此操作也称为并发插入。

需要注意的是，并发插入的数据不能是 DELETE 操作删除的记录，并且只能在表中最后的一行记录后继续增加新记录。

（4）设置“显式”写的表级锁。

```
#① 在客户端 A 中添加表级写锁
mysql> LOCK TABLE mydb.table_lock WRITE;
Query OK, 0 rows affected (0.00 sec)
#② 在客户端 A 中执行更新和查询操作
mysql> UPDATE mydb.table_lock SET id=1 WHERE id=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> SELECT * FROM mydb.table_lock \G
*************************** 1. row ***************************
id: 3
*************************** 2. row ***************************
id: 1
*************************** 3. row ***************************
id: 4
3 rows in set (0.00 sec)
#③ 在客户端 B 中执行查询操作
```

```
mysql> SELECT * FROM mydb.table_lock;
#此处光标会不停闪烁,进入锁等待状态
```

从上述操作可以看出,添加了写锁的用户,可以执行读/写操作(如增删改查),而其他用户不论执行任何操作(如 SELECT),都只能处于等待状态,直到写锁被释放,才能够执行。

11.3.3　行级锁

InnoDB 存储引擎的锁机制相对于 MyISAM 存储引擎的锁复杂一些,原因在于它既有表级锁又有行级锁。其中,InnoDB 表级锁的应用与 MyISAM 表级锁的相同,这里不再赘述。那么 InnoDB 存储引擎的表什么时候添加表级锁,什么时候添加行级锁呢? 只有通过索引条件检索的数据 InnoDB 存储引擎才会使用行级锁,否则将使用表级锁。

InnoDB 的行级锁根据操作的不同也分为共享锁和排他锁。为了读者更好地理解,下面以“隐式”的行级锁和“显式”的行级锁为例进行详细讲解。

1. “隐式”行级锁

当用户对 InnoDB 存储引擎表执行 INSERT、UPDATE、DELETE 等写操作前,服务器会“自动”地为通过索引条件检索的记录添加行级排他锁;直到操作语句执行完毕,服务器再“自动”地为其解锁。

而语句的执行时间可以看作是“隐式”行级锁的生命周期,且该生命周期的持续时间一般都比较短暂。通常情况下,若要增加行级锁的生命周期,最常使用的方式是事务处理,让其在事务提交或回滚后再释放行级锁,使行级锁的生命周期与事务的相同。

为了读者更好地理解,下面在事务中演示“隐式”行级锁的使用。具体步骤如下。

(1) 创建 InnoDB 表并插入测试数据。

```
mysql> CREATE TABLE mydb.row_lock (
    ->    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    ->    name VARCHAR(60) NOT NULL,
    ->    cid INT UNSIGNED,
    ->    KEY cid (cid)
    -> )DEFAULT CHARSET=utf8;
Query OK, 0 rows affected (0.01 sec)
mysql> INSERT INTO mydb.row_lock (name, cid) VALUES ('铅笔', 3),
    -> ('风扇', 6), ('绿萝', 1), ('书包', 9), ('纸巾', 20);
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

(2) 设置“隐式”行级锁。

打开两个客户端 A 和 B,在客户端 A 中为 mydb.row_lock 设置“隐式”行级的排他锁后,然后在客户端 B 中执行 SELECT 和 DELETE 操作。具体 SQL 语句及执行结果如下。

```
#① 在客户端 A 中,修改 cid 等于 3 的 name 值
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> UPDATE mydb.row_lock SET name = 'cc' WHERE cid = 3;
```

```
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
#② 在客户端 B 中,删除 cid 等于 2 和 3 的记录
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> DELETE FROM mydb.row_lock WHERE cid = 2;
Query OK, 0 rows affected (0.00 sec)
mysql> DELETE FROM mydb.row_lock WHERE cid = 3;
#此处光标会不停闪烁,进入锁等待状态
#③ 在客户端 A 和 B 中,回滚以上的操作
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
```

从以上执行结果可知,一个客户端对 InnoDB 表执行 UPDATE 操作时,对符合索引条件的记录会隐式地添加一个行级锁,与此同时其他用户不能再执行写操作,但可以操作不符合索引条件的记录(如删除 cid 等于 2 的记录)。

2. "显式"行级锁

对于 InnoDB 表来说,若要保证当前事务中查询出的数据不会被其他事务更新或删除,利用普通的 SELECT 语句是无法办到的,此时需要利用 MySQL 提供的"锁定读取"的方式为查询操作显式地添加行级锁。其基本语法格式如下。

```
SELECT 语句 FOR UPDATE|LOCK IN SHARE MODE
```

在上述语法中,只需在正常的 SELECT 语句后添加 FOR UPDATE 或 LOCK IN SHARE MODE 即可实现"锁定读取",前者表示在查询时添加行级排他锁,后者表示在查询时添加行级共享锁。

用户在向 InnoDB 表显式添加行级锁时,InnoDB 存储引擎首先会"自动"地向此表添加一个意向锁,然后再添加行级锁。此意向锁是一个隐式的表级锁,多个意向锁之间不会产生冲突且互相兼容。意向锁是由 MySQL 服务器根据行级锁是共享锁还是排他锁,自动添加意向共享锁或意向排他锁,不能人为干预。

意向锁的作用就是标识表中的某些记录正在被锁定或其他用户将要锁定表中的某些记录。相对行级锁,意向锁的锁定粒度更大,用于在行级锁中添加表级锁时判断它们之间是否能够互相兼容。好处就是大大节约了存储引擎对锁处理的性能,更加方便地解决了行级锁与表级锁之间的冲突。

为了读者更好地理解,下面通过一个表格展示表级的共享/排他锁与意向共享/排他锁之间的兼容性关系,具体如表 11-6 所示。

表 11-6 表级共享/排他锁与意向共享/排他锁之间的关系

	表级共享锁	表级排他锁	意向共享锁	意向排他锁
表级共享锁	兼容	冲突	兼容	冲突
表级排他锁	冲突	冲突	冲突	冲突

续表

	表级共享锁	表级排他锁	意向共享锁	意向排他锁
意向共享锁	兼容	冲突	兼容	兼容
意向排他锁	冲突	冲突	兼容	兼容

需要注意的是，InnoDB 表中当前用户的意向锁若与其他用户要添加的表级锁冲突时，有可能会发生死锁而产生错误。

接下来利用上面创建的 mydb.row_lock 表，演示添加行级排他锁时客户端 A 和客户端 B 执行 SQL 语句的状态，具体步骤如下。

```
#① 在客户端 A 中，为 cid 等于 3 的记录添加行级排他锁
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM mydb.row_lock WHERE cid=3 FOR UPDATE;
+----+------+------+
| id | name | cid  |
+----+------+------+
| 1  | 铅笔 |   3  |
+----+------+------+
1 row in set (0.00 sec)
#② 在客户端 B 中，为 cid 等于 2 的记录添加隐式行级排他锁，设置表级排他锁
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> UPDATE mydb.row_lock SET name='lili' WHERE cid=2;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0
mysql> LOCK TABLE mydb.row_lock READ;
#此处光标会不停闪烁，进入锁等待状态
#③ 回滚以上的操作并释放表级锁
```

从以上的执行结果可知，在客户端 A 中为 cid 等于 3 的记录添加行级排他锁后，在客户端 B 中，可以为除 cid 等于 3 外的记录添加行级排他锁（如 cid 等于 2 的隐式排他锁），但是在为表添加表级共享锁时会发生冲突，进行锁等待状态。

此外，默认的情况下，当 InnoDB 处于 REPEATABLE READ（可重复读）的隔离级别时，行级锁实际上是一个 next-key 锁，它是由间隙锁（gap lock）和记录锁（record lock）组成的。其中，记录锁（record lock）就是前面讲解的行锁；间隙锁指的是在记录索引之间的间隙、负无穷到第 1 个索引记录之间或最后 1 个索引记录到正无穷之间添加的锁，它的作用就是在并发时防止其他事务在间隙插入记录，解决了事务幻读的问题。

为了读者更好地理解，下面为 mydb.row_lock 表添加行锁，查看间隙锁是否存在。具体步骤如下。

```
#① 在客户端 A 中，为 cid 等于 3 的记录添加行锁
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM mydb.row_lock WHERE cid=3 FOR UPDATE;
```

```
+----+------+------+
| id  | name   | cid   |
+----+------+------+
|  1  | 铅笔    |   3   |
+----+------+------+
1 row in set (0.00 sec)
#② 在客户端 B 中,插入 cid 等于 1、2、5、6 的记录
mysql> INSERT INTO mydb.row_lock(name, cid) VALUES('电视', 1);
#此处光标会不停闪烁,进入锁等待状态
mysql> INSERT INTO mydb.row_lock(name, cid) VALUES('电视', 2);
#此处光标会不停闪烁,进入锁等待状态
mysql> INSERT INTO mydb.row_lock(name, cid) VALUES('电视', 5);
#此处光标会不停闪烁,进入锁等待状态
mysql> INSERT INTO mydb.row_lock(name, cid) VALUES('电视', 6);
Query OK, 1 row affected (0.00 sec)
```

在上述操作中,客户端 A 在 cid 等于 3 的记录中添加了行锁,理论上其他用户在并发时可以插入除 cid 等于 3 的任意记录,但是因为间隙锁的存在,服务器也会锁定当前表中 cid(值分别为 1、3、6、9、20)值为 3 的记录左右的间隙,间隙的区间范围为[1,3)和[3,6)。

值得一提的是,在执行 SELECT…FOR UPDATE 时,若检索时未使用索引,则 InnoDB 存储引擎会给全表添加一个表级锁,并发时不允许其他用户进行插入。另外,若查询条件使用的是单字段的唯一性索引,InnoDB 存储引擎的行级锁不会设置间隙锁。

间隙锁的使用虽然解决了事务幻读的情况,但是也会造成行锁定的范围变大,若在开发时想要禁止间隙锁的使用,可以将事务的隔离级别更改为 READ COMMITTED(读取提交)。

多学一招:查看 InnoDB 表的锁

InnoDB 存储引擎的锁比较复杂,读者可以在添加一个行锁后,使用 SHOW ENGINE INNODB STATUS 语句查看当前表中添加的锁的类型。另外,在查看时要保证开启系统变量 innodb_status_output_locks 才能获取锁定的信息。例如,查看 mydb.row_lock 表添加的锁,部分信息如下。

```
TABLE LOCK table `mydb`.`row_lock` trx id 10386 lock mode IX
RECORD LOCKS space id 247 page no 4 n bits 80 index cid of table `mydb`.`row_lock` trx id 10386 lock_mode X
⋮ (此处省略部分内容)
RECORD LOCKS space id 247 page no 3 n bits 80 index PRIMARY of table `mydb`.`row_lock` trx id 10386 lock_
mode X locks rec but not gap
⋮ (此处省略部分内容)
RECORD LOCKS space id 247 page no 4 n bits 80 index cid of table ``mydb`.`row_lock` trx id 10386 lock_mode
X locks gap before rec
⋮ (此处省略部分内容)
```

在上述信息中,"IX"表示 mydb.row_lock 中添加了一个意向排他锁,"X"表示 next-key lock 的排他锁,"X locks rec but not gap"表示记录锁,"X locks gap before rec"表示间隙锁。它们之间的关系为"IX"在"X"之前添加,而"X"是由"X locks rec but not gap"和"X locks gap before rec"组成的。

11.4　分表技术

通常情况下,项目中数据库的数据随着时间的推移会越来越多,而单张数据表存储的数据又是有限的,当其达到一定的量级(如百万级)时,即使添加了索引,执行查询操作依然会变慢,特别是在并发操作时,会增加单表的访问压力。此时,可以考虑使用分表技术,将单张数据表根据不同的需求进行拆分,从而达到分散单表压力的目的,提升数据库的访问性能。

MySQL 中常用的分表技术有两种,分别为水平分表和垂直分表。所谓水平分表指的是将一张数据表中的全部记录分别存储到多张数据表中,因此水平分表在创建时,必须保证各数据表涉及的字段全部相同。所谓垂直分表指的是将同一个业务的不同字段分别存储到多张数据表中,因此垂直分表在创建时,各数据表仅通过一个字段进行连接,其他字段都不相同。

为了读者更好地理解,接下来分别讲解水平分表和垂直分表的实现原理以及各自的优缺点。

1. 水平分表

水平分表是一种物理创建表的设计,它是用户根据指定的需求将记录分别存储到各个分表中,每张数据表的字段相同,但是名称不同。通常情况下,就是根据表的名称对分表进行增删改查操作,如图 11-1 所示。

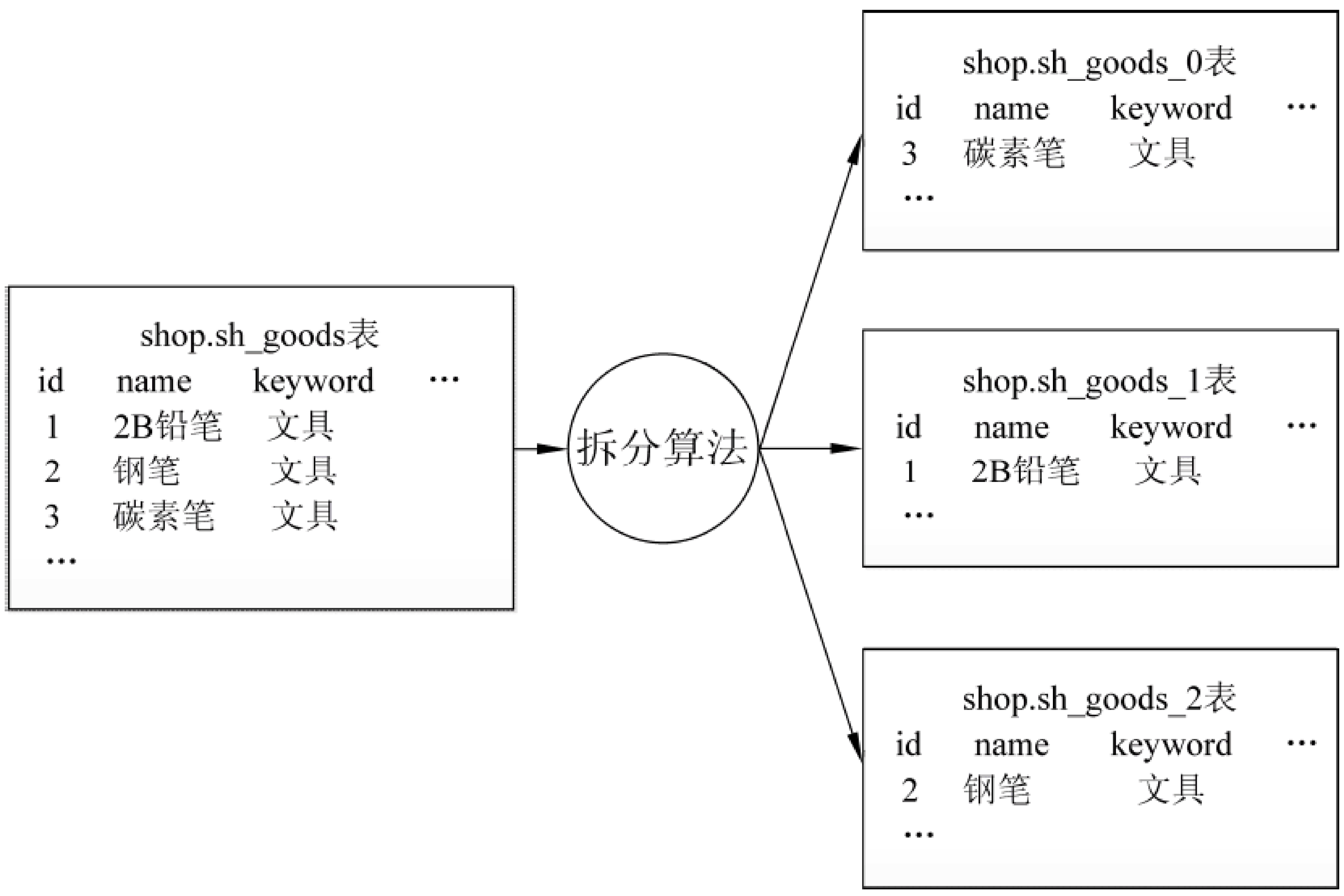


图 11-1　水平分表

在图 11-1 中,水平分表的拆分方式(算法)可以有多种,根据项目的业务不同可以演化出多种不同的方式。最常使用的方式就是利用记录 ID 与分表个数取余获取分表的编号,然后再对获取的分表执行指定的操作。

例如,对 sh_goods 表进行水平分表,在图 11-1 中设置了 3 个分表,因此根据 id%3 获取的余数将对应的记录插入到对应的分表中,同理在对指定记录进行删除、修改、查询时,也是

利用以上取余的方式到指定的分表中进行相关的操作。

除此之外，水平分表的拆分方式还可以根据商品的创建时间、品牌、店铺、销量等级等的不同将其分别存储到不同的分表中。而分表的数量以及拆分方式还需考虑表的预估容量、可扩展性等因素具体去设计。

总结：水平分表使单张表的数据能够保持在一定的量级，在操作时又因其表结构完全相同，只需增加获取对应分表名称的运算，就可以提高系统的稳定性和负载能力。但同时它也有一定的缺点，水平分表使得数据分散存储，加大了数据的维护难度。

2. 垂直分表

设计数据表时，若一个表中含有很多字段，其中有一部分字段经常被使用，而有一部分字段不常被使用，那么在对此数据表进行操作时，不常用的字段也会占据一定的资源，会对系统的整体性能造成一定的干扰和影响。

为了减少资源的开销、提升运行效率，可以采用垂直分表的方式，将数据表中的字段根据使用的频率分别存储到不同的表中。

假如有一个用户信息表，它含有 12 个字段，分别为用户 ID、用户名、密码、邮箱、手机号、QQ、是否激活、用户级别、性别、注册时间、创建时间和更新时间。其中，只有前 7 个字段会在用户登录时经常用到，其余字段的使用率则很小，这时就可以利用垂直分表的设计方式将其拆分成一个主表和一个从表，如表 11-7 所示。

表 11-7　垂直分表

主 表 字 段	从 表 字 段
用户 ID（主键）	用户 ID（主键）
用户名	用户级别
密码	性别
邮箱	注册时间
手机号	创建时间
QQ	更新时间
是否激活	—

在表 11-7 中，主表和从表利用用户 ID 进行连接即可获取一个用户的完整信息。当不需要完整信息时，只需要对相应的表进行操作即可。

总结：垂直分表后业务逻辑更加清晰，方便数据进行整合与扩展，还可以根据实际需求实现动静分离，为各分表选择不同的存储引擎（如查询操作多可以使用 MyISAM 等）。但同时它也有一定的缺点，需要管理冗余字段、查询所有数据需要进行连接。

11.5　分区技术

11.5.1　分区概述

对于单表数据量过大的问题，除了可以使用分表技术，在物理上创建多张数据表解决

外,还可以使用 MySQL 本身支持的分区技术提高数据库的整体性能。

所谓分区技术,就是在操作数据表时可以根据给定的算法,将数据在逻辑上分到多个区域中存储。此外,在分区中还可以设置子分区,将数据存放到更加具体的区域内。比如大量的水果(数据)可以分别存储在多个仓库(分区)中,在仓库中又可以划分出固定的区域(子分区)用来存放不同种类的水果(数据)。

分区技术可以使一张数据表中的数据存储在不同的物理磁盘中,相比单个磁盘或文件系统能够存储更多的数据,实现更高的查询吞吐量。若在 WHERE 子句中包含分区条件,系统只需扫描相关的一个或多个分区而不用全表扫描,从而提高查询效率。

MySQL 中分区技术在使用时对存储引擎以及锁有一定的要求,具体内容如下。

(1) 分区技术不适用于 MERGE、CSV 或 FEDERATED 存储引擎。

(2) InnoDB 分区表不能设置外键,同样的,与外键相关的主表和从表也不能被分区。

(3) MySQL 5.7 中 InnoDB 表不支持子分区在多个磁盘中存储,目前仅有 MyISAM 存储引擎支持。

(4) 同一个分区表的所有分区必须使用相同存储引擎。当建表时未指定存储引擎,在创建分区时必须设置存储引擎。

(5) MySQL 5.6 及更早的版本中,对分区的 MyISAM 表进行操作时,会锁定所有的分区,直到操作完成后才会释放锁。而在 MySQL 5.7 中,仅会锁定与操作相关的分区,不会影响其他分区。

11.5.2 分区管理

在了解了分区的作用以及特点后,本节将对创建分区、增加分区以及删除分区的操作进行详细讲解。

1. 创建分区

在创建数据表时,可同时完成分区的创建,其基本语法格式如下。

```
CREATE TABLE 数据表名称
[(字段与索引列表)][表选项]
PARTITION BY 分区算法 (分区字段)[PARTITIONS 分区数量]
[SUBPARTITION BY 子分区算法 (子分区字段)[SUBPARTITIONS 子分区数量]]
[(
PARTITION 分区名 [VALUES 值][其他选项][(SUBPARTITION 子分区名 [其他选项])],
⋮
)];
```

在上述语法中,分区是在表选项后添加 PARTITION BY 实现,一个表最多仅可以创建 1024 个分区。其中,分区算法有 4 种,分别为 LIST、RANGE、HASH 和 KEY,每种算法对应的分区字段不同,具体语法如下。

```
RANGE/LIST (表达式) 或 COLUMNS(字段列表)
HASH(表达式)
KEY[ALGORITHM={1|2}](字段列表)
```

在上述语法中，KEY 算法的 ALGORITHM 选项用于指定 key-hashing 函数的算法，其值等于 1，适用于 MySQL 5.1；默认值是 2，适用于 MySQL 5.5 及以后版本。此外，子分区算法仅支持 HASH 和 KEY。

在指定分区算法和分区字段后，RANGE 和 LIST 分区必须用 PARTITION…VALUES 具体定义每个分区选项，且只有这两个算法有 VALUES 选项。具体语法如下。

```
#RANGE 算法必须使用 LESS THAN 而不能用 IN
PARTITION 分区名 VALUES LESS THAN {(表达式 | 值列表) | MAXVALUE}
#LIST 算法必须使用 IN 而不能用 LESS THAN
PARTITION 分区名 VALUES IN (值列表)
```

在上述语法中，分区名要符合 MySQL 标识符的规则。但需要注意的是，分区名称不区分大小写。分区的其他选项如表 11-8 所示。

表 11-8　分区其他选项

选　项　名	描　　述
ENGINE	用于设置分区的存储引擎
COMMENT	用于为分区添加注释
DATA DIRECTORY	用于为分区设置数据目录
INDEX DIRECTORY	用于为分区设置索引目录
MAX_ROWS	用于为分区设置最大的记录数
MIN_ROWS	用于为分区设置最小的记录数
TABLESPACE	用于为分区设置表空间名称

为了读者更好地理解，下面以创建 LIST 和 HASH 分区为例进行演示，具体 SQL 语句及执行结果如下。

(1) 创建 LIST 分区。

```
mysql>CREATE TABLE mydb.p_list(
    ->    id INT AUTO_INCREMENT COMMENT 'ID 编号',
    ->    name VARCHAR(50) COMMENT '姓名',
    ->    dpt INT COMMENT '部门编号',
    ->    KEY(id)
    ->)ENGINE=INNODB
    ->PARTITION BY LIST(dpt)(
    ->    PARTITION p1 VALUES IN (1, 3),
    ->    PARTITION p2 VALUES IN (2, 4)
    ->);
Query OK, 0 rows affected (0.03 sec)
```

在上述语句中，以 mydb.p_list 表中的 dpt 字段进行分区，当该字段的值为 1 或 3 时，将对应的记录放在名为 p1 的分区中；当该字段的值为 2 或 4 时，将对应的记录放在名为 p2 的分区中。分区创建完成后，会在 MySQL 的数据文件 data/mydb 目录下看到对应的分区数据文件，如下所示。

```
p_list#p#p1.idb
p_list#p#p2.idb
```

上述的文件名称中,p_list 是建立分区的数据表名,p1 和 p2 表示分区的名称。

此外,读者还可以利用 SHOW CREATE TABLE 查看分区的创建语句,使用 SHOW TABLE STATUS 查看指定数据库下对应数据表是否有创建分区,若有则 Create_options 字段的值为 partitioned;在执行 SQL 语句时可以使用 EXPLAIN 进行分析。

需要注意的是,在使用分区时,只有 WHERE 子句后字段是分区字段,SQL 操作才会用到分区。

(2) 创建 HASH 分区。

```
mysql>CREATE TABLE mydb.p_hash(
    ->    id INT AUTO_INCREMENT,
    ->    name VARCHAR(50),
    ->    dpt INT,
    ->    KEY(id)
    ->)ENGINE=INNODB
    ->PARTITION BY HASH(dpt)PARTITIONS 3;
Query OK, 0 rows affected (0.04 sec)
```

上述语句中,使用 HASH 算法为 p_hash 表创建了 3 个分区,分区文件的序号默认从 0 开始,当有多个分区时依次递增加 1。例如,以上创建的分区序号依次为 0、1 和 2。

小提示:

(1) 当创建分区的表时,主键必须包含在建立分区的字段中。

(2) 当创建分区的表仅有一个 AUTO_INCREMENT 字段时,该字段必须为索引字段。

2. 增加分区

在操作时,若数据表没有创建分区,则可以利用以下的语法为已经创建的数据表添加分区,其基本语法如下。

```
ALTER TABLE 数据表名称 PARTITION BY 分区算法…;
```

在上述语法中,PARTITION BY 后可以添加的内容与 CREATE TABLE 的相同。此外,对已经含有分区的数据表再添加分区时,则可以使用以下的语法。

```
#LIST 或 RANGE 分区
ALTER TABLE 数据表名称 ADD PARTITION (分区选项, …);
#HASH 或 KEY 分区
ALTER TABLE 数据表名称 PARTITIONS 数量;
```

在上述语法中,分区选项与 CREATE TABLE 中创建分区的选项相同。为了读者更好地理解,下面为 mydb 数据库下 p_list 和 p_hast 数据表添加分区。具体 SQL 语句及执行结果如下。

```
mysql>ALTER TABLE mydb.p_list ADD PARTITION (
    ->    PARTITION new1 values IN (5, 6),
```

```
    -> PARTITION new2 values IN (7, 8)
    ->);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql>ALTER TABLE mydb.p_hash ADD PARTITION PARTITIONS 1;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

分区添加完成后，读者可按上面讲解的方式查看分区。值得一提的是，当添加分区的数据表已经含有数据时，会按照分区的算法将已有的数据分配到不同的分区中。

3. 删除分区

在分区管理过程中，若某个表不再需要设置分区时，可以通过 MySQL 提供的方式进行删除，但不同算法的分区删除方式不相同。其基本语法格式如下。

```
#删除 HASH、KEY 分区
ALTER TABLE 数据表名称  COALESCE PARTITION 数量;
#删除 RANGE、LIST 分区
ALTER TABLE 数据表名称 DROP PARTITION 分区名称;
```

在上述语法中，HASH 与 KEY 算法的分区在删除时，会将该分区内的数据重新整合到剩余的分区中，而 RANGE 与 LIST 算法的分区在删除时会同时删除分区中保存的数据。此外，当数据表的分区仅剩一个时，不能通过以上的方式删除，只能利用 DROP TABLE 的方式删除表。

为了读者更好地理解，下面以删除 mydb. p_list 表中的分区为例进行演示，具体 SQL 语句及执行结果如下。

（1）添加测试数据，用于测试删除分区后数据的变化。

```
mysql>INSERT INTO mydb.p_list(name, dpt) VALUES
    ->('Tom', 5), ('Lucy', 6), ('Lily', 7), ('Jim', 8);
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

在上述语句中，根据添加分区的设置，dpt 为 5 和 6 的记录保存在名为 new1 的分区中，dpt 为 7 和 8 的记录保存在名为 new2 的分区中。

（2）删除 mydb. p_list 表中名为 new1 的分区。

```
mysql>ALTER TABLE mydb.p_list DROP PARTITION new1;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

按照以上操作删除分区后，可以到 data/mydb 目录下看到名为 p_list#p#new1. ibd 的分区数据文件已被删除，然后利用以下语句可以看出当前 mydb. p_list 表中 new1 分区下保存的数据也被同时删除了，仅剩 new2 分区下的两条记录。如下所示。

```
mysql>SELECT * FROM mydb.p_list;
+----+------+-------+
| id | name | dpt   |
+----+------+-------+
|  3 | Lily |   7   |
|  4 | Jim  |   8   |
+----+------+-------+
2 rows in set (0.00 sec)
```

值得一提的是，若在开发中仅要清空各分区表中的数据，不删除对应的分区文件，可以使用以下的语句实现。其基本语法如下。

```
ALTER TABLE 数据表名称 TRUNCATE PARTITION {分区名称 | ALL}
```

在上述语法适用于所有算法的分区，当要删除表中所有分区中保存的数据时，使用 ALL 代替即可；若要删除指定的分区，则需要使用分区名称，多个分区之间使用逗号(,)分隔。

11.6　数据碎片与维护

在 MySQL 数据库中，DELETE 删除一条记录时，仅仅删除了数据表中保存的数据，而记录占用的存储空间会被保留。因此，长期删除数据、添加数据的过程中，索引文件和数据文件都将产生“空洞”，形成很多不连续的碎片，造成数据表占用的空间变大，但是表中的记录数却很少的情况发生。

若要解决以上数据碎片造成的影响，可使用 MySQL 提供的方式 OPTIMIZE TABLE (支持 MySQL 中常见的存储引擎 MyISAM 和 InnoDB)重新组织表中数据和关联索引数据的物理存储，减少存储空间并提高访问表时的 I/O 效率。

为了读者更好地理解，下面通过一个案例进行简单的演示。

(1) 创建数据表、添加测试数据。

要想看出数据碎片整理与未整理，需要在数据表中添加大量的数据，然后直接查看其数据文件大小的变化。下面利用数据复制的方式为新建的数据表 mydb. my_optimize 添加测试数据(如此处添加到 50 万条以上的数据)。具体 SQL 语句及执行结果如下。

```
#① 创建数据表
mysql>CREATE TABLE mydb.my_optimize (
    ->   id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    ->   name VARCHAR(20) NOT NULL DEFAULT ''
    ->);
Query OK, 0 rows affected (0.01 sec)
#② 添加测试数据
mysql>INSERT INTO mydb.my_optimize(name)
    ->VALUES ('TOM'), ('JIMMY'), ('LUCK'), ('CAKE');
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

```
#③ 数据复制添加测试数据
mysql> INSERT INTO mydb.my_optimize(name) SELECT name FROM mydb.my_optimize;
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0
#多次执行③中的语句直到数据达到 50 万条以上,此处省略
```

(2) DELETE 数据,查看数据存储文件的大小。

在删除数据前,打开数据库 data 目录,查看添加完数据后 my_optimize.idb 的大小,大约为 40MB。然后执行以下的 DELETE 操作,具体 SQL 语句及执行结果如下。

```
mysql> DELETE FROM mydb.my_optimize WHERE name = 'LUCK';
Query OK, 262144 rows affected (2.25 sec)
```

数据删除后,再次查看 my_optimize.idb 会发现数据索引文件的大小并没有变化,而此时 mydb.my_optimize 表中已经删除了所有名为 LUCK 的记录。

(3) 整理数据,查看数据存储文件的大小。

为了整理以上删除操作产生的数据碎片,接下来使用 OPTIMIZE TABLE 进行碎片整理。具体 SQL 语句及执行结果如下。

```
mysql> OPTIMIZE TABLE mydb.my_optimize \G
*************************** 1. row ***************************
   Table: mydb.my_optimize
      Op: optimize
Msg_type: note
Msg_text: Table does not support optimize, doing recreate + analyze instead
*************************** 2. row ***************************
   Table: mydb.my_optimize
      Op: optimize
Msg_type: status
Msg_text: OK
2 rows in set (1.10 sec)
```

在上述执行结果中,InnoDB 存储引擎的数据表不支持 OPTIMIZE TABLE 操作,因此给出第一条记录进行报错。然后系统自动使用 ALTER TABLE…FORCE 语句重新构建表并整理相关的数据碎片,释放未使用的存储空间,返回第 2 条记录信息。其中,Op 表示执行 optimize 操作,Msg_type 表示信息的类型,除此之外还有 error、info 和 warning;Msg_text 表示具体的返回信息内容。

完成上述操作后,再次查看 my_optimize.idb 会发现数据索引文件的大小变为 31MB 左右。可以清晰地看出在对数据表进行维护后,解决了因数据碎片产生的空间的浪费以及查询速度慢的问题。

除了以上讲解的 OPTIMIZE 操作外,还可以使用 ALTER TABLE 将数据表的存储引擎修改为当前数据表的存储引擎,实现对数据碎片的整理。例如,上面的步骤(3)可以使用以下语句代替。

```
ALTER TABLE mydb.my_optimize ENGINE= 'InnoDB';
```

需要注意的是，修复数据表的数据及索引碎片时，会把所有的数据文件重新整理一遍。因此，若数据表的记录数比较大，也会消耗一定的资源，所以不能频繁地对数据碎片进行维护，可根据实际的情况按周、月或季度等进行操作。

11.7　动手实践：数据库优化实战

数据库的学习在于多看、多学、多想、多动手，只有将理论与实际相结合，才能够体现出数据开发与管理的重要性，展现知识学习的价值与力量。接下来请结合本章所学的知识完成数据库的优化操作。

【实践目标】

此实践的目标就是能够根据文字提示，完成百万级数据常规的优化操作。

【实践需求】

（1）在 mydb 数据库中创建 my_user 用户表并添加两百万条数据用于测试。my_user 数据表的字段有 id、name 和 pid。

（2）开启慢查询日志，获取查询时间超过 0.5 秒的查询语句信息。

（3）开启 profile 机制，获取语句执行的精确时间。

（4）为 pid 添加索引，优化查询语句，增强查询效率。

（5）优化 LIMIT 分页查询的效率。

（6）启动查询缓存功能，优化查询效率。

【动手实践】

1. 创建测试数据

在 mydb 数据库中创建一个用户表 my_user，具体 SQL 语句及执行结果如下。

```
mysql>CREATE TABLE IF NOT EXISTS mydb.my_user(
    ->    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT COMMENT '商品编号',
    ->    name VARCHAR(120) NOT NULL COMMENT '用户名',
    ->    pid INT UNSIGNED NOT NULL COMMENT '部门编号'
    ->)DEFAULT CHARSET=utf8;
Query OK, 0 rows affected (0.01 sec)
```

为了便于测试数据库优化的效果，下面通过自定义函数和存储过程的方式为 my_user 表添加百万级别的数据，具体 SQL 语句及执行结果如下。

```
mysql>DROP FUNCTION IF EXISTS mydb.rand_str;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql>DELIMITER $$
mysql>CREATE FUNCTION mydb.rand_str(len INT) RETURNS VARCHAR(50)
    ->BEGIN
    ->    DECLARE chars_str VARCHAR(100) DEFAULT 'abcdefghijklmnopqrstuvwx
yzABCDEFGHIJKLMNOPQRSTUVWXYZ';
```

```
        -> DECLARE res VARCHAR(50) DEFAULT '';
        -> DECLARE i INT DEFAULT 0;
        -> WHILE i < len DO
        ->   SET res = CONCAT(res, SUBSTRING(chars_str, FLOOR(1 + RAND() * 52), 1));
        ->   SET i=i+1;
        -> END while;
        -> RETURN res;
        -> END
        -> $$
Query OK, 0 rows affected (0.00 sec)
mysql> DELIMITER;
mysql> DROP PROCEDURE IF EXISTS mydb.my_user_pro;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> DELIMITER $$
mysql> CREATE PROCEDURE mydb.my_user_pro (in max_num INT)
        -> BEGIN
        -> DECLARE i INT DEFAULT 0;
        -> REPEAT
        ->   SET i=i+1;
        ->   INSERT INTO mydb.my_user(name, pid) VALUES
        ->   (rand_str(5), FLOOR(1 + RAND() * 10));
        -> UNTIL i=max_num END REPEAT;
        -> END
        -> $$
Query OK, 0 rows affected (0.00 sec)
mysql> DELIMITER;
```

在上述语句中，在创建函数与存储过程前首先执行删除操作，避免 mydb 数据库含有相同的函数与存储过程。自定义的 rand_str()函数可以获取指定位数的字符串，my_user_pro()存储过程可以为 my_user 表完成指定数量数据的添加。

下面调用 my_user_pro()存储过程，为 my_user 表添加两百万条数据，具体 SQL 语句及执行结果如下。

```
mysql> CALL mydb.my_user_pro(2000000);
Query OK, 1 row affected (2 min 49.76 sec)
```

2. 开启慢查询日志

慢查询日志是 MySQL 提供的一种记录所有执行时间超过指定时间界限的 SQL 语句，然后可以根据写入日志内的语句对 MySQL 进行分析优化。

默认情况下，没有开启慢查询日志，下面打开 MySQL 的 my.ini 配置文件，开启慢查询日志功能，同时设置慢查询日志文件的路径以及时间界限。具体配置如下。

```
slow_query_log=on
slow_query_log_file="C:/slow.log"
long_query_time = 0.5
```

在上述配置中，slow_query_log 用于开启慢查询日志，slow_query_log_file 用于设置日志文件保存的路径，默认保存到 MySQL 的 data 目录下，这里将其保存到 C 盘下 slow.log 文件中。long_query_time 表示查询时间超出指定的时间（如 0.5 秒）就将其当做慢查询，将其保存到 slow.log 文件中，默认的时间为 10 秒。

完成设置后，重新启动 MySQL 使配置生效。然后执行以下 SQL 语句，并打开 C:\slow.log 文件，分析慢查询日志中的信息。具体如下。

```
#① 在客户端中执行以下 SQL 语句
SELECT COUNT(*) FROM my_user WHERE pid=2;
#② 使用文本编辑器(如记事本、Notepad++)打开 C:\slow.log 文件,查看保存的日志信息
mysql5.7.22, Version: 5.7.22-log (MySQL Community Server (GPL)). started with:
TCP Port: 3306, Named Pipe: (null)
Time                 Id Command    Argument
#Time: 2018-07-24T08:30:25.532616Z
#User@Host: root[root] @localhost [::1]  Id:     2
#Query_time: 0.783045  Lock_time: 0.000000 Rows_sent: 1  Rows_examined: 2000000
SET timestamp=1532421025;
select count(*) from test.my_user where pid=2;
```

以上的日志信息中，Time 表示执行慢查询语句的服务器时间，User@Host 指定执行慢查询语句的用户，Query_time 表示慢查询语句的执行时间，Lock_time 表示锁定的时间，Rows_sent 表示发送的记录数，Rows_examined 表示检索的记录数，SET timestamp 是将信息写入日志的时间戳，最后一句是慢查询的 SQL 语句。

值得一提的是，虽然慢查询日志是 MySQL 优化及调试的一个重要工具，但是开启后慢查询会占用一定的系统资源与空间。因此，建议在项目开发阶段可以开启用于数据库调试与优化，在项目上线后要将其关闭。

3. 开启记录查询的精确时间功能

在优化数据操作时，若仅想获取精确到毫秒查询时间，可以启动 MySQL 提供的 profile 机制，它会记录每次操作的具体时间（精度为小数点后 8 位）。

下面开启记录查询的精确时间功能，查看 SQL 语句的执行时间。具体 SQL 语句及执行结果如下。

```
#① 开启 profile 机制
mysql>SET profiling=1;
Query OK, 0 rows affected, 1 warning (0.00 sec)
#② 执行查询语句
mysql>SELECT COUNT(*) FROM mydb.my_user WHERE pid=2;
+----------+
| count(*) |
+----------+
|  199829  |
+----------+
1 row in set (0.59 sec)
#③ 获取精确时间
```

```
mysql> SHOW PROFILES\G
*************************** 1. row ***************************
Query_ID: 1
Duration: 0.58827200
   Query: SELECT COUNT(*) FROM mydb.my_user WHERE pid=2
1 row in set, 1 warning (0.00 sec)
```

从以上的操作可以看出，执行查询语句时，下面的显示时间0.59秒为近似值，而SHOW PROFILES可以看到它的精确值为0.58827200。需要注意的是，为了保证数据库更好的性能，在不需要分析SQL语句执行时间时，最好利用SET profiling=0关闭profile机制。

4. 添加索引

通过步骤2和步骤3的分析可以看出，在my_user表百万级数据量时，查询pid等于2的所有员工记录数花费的时间约为0.59秒，此时可以通过为pid添加索引来加快查询的速度。具体步骤如下。

```
#① 在添加索引前,查看一下 my_user 表的数据索引文件的大小大概为 76MB
#② 为 my_user 表的 pid 字段添加一个普通索引
mysql> ALTER TABLE mydb.my_user ADD INDEX(pid);
Query OK, 0 rows affected (5.51 sec)
Records: 0  Duplicates: 0  Warnings: 0
#③ 添加索引后,查看一下 my_user 表的数据索引文件的大小大概为 108MB
#④ 添加索引后查询 pid 等于 2 的员工数据量
mysql> SELECT COUNT(*) FROM mydb.my_user WHERE pid=2;
+----------+
| count(*) |
+----------+
|   199829 |
+----------+
1 row in set (0.08 sec)
```

从上述的操作可知，在添加索引前，查询pid等于2的所有员工记录数花费的时间约为0.59秒，数据索引文件大小大概为76MB；而添加索引后，相同的SQL语句查询的时间约为0.08秒，文件大小大概为108MB。明显可以看出，添加索引后，SQL语句的执行速度提高了7倍多，但是数据索引文件的大小也比未添加索引前增加了大约1.4倍。因此，在实际开发时，要均衡使用索引带来的优势与劣势。

5. LIMIT分页优化

在实际应用中，影响查询速度的因素除了是否添加索引外，获取记录的页数也会影响查询的效率。例如，若每次获取my_user表中10条记录，则获取第8999页、89999页数据所花费的时间如下所示。

```
#① 执行以下 SQL 语句
SELECT * FROM mydb.my_user LIMIT 90000, 10;
SELECT * FROM mydb.my_user LIMIT 900000, 10;
#② 利用 profile 机制获取精确的查询时间
mysql> SHOW PROFILES;
+----------+------------+------------------------------------------------+
| Query_ID | Duration   | Query                                          |
+----------+------------+------------------------------------------------+
: (此处省略其他的显示记录)
|        6 | 0.03255100 | SELECT * FROM mydb.my_user LIMIT 90000, 10     |
|        7 | 0.34337050 | SELECT * FROM mydb.my_user LIMIT 900000, 10    |
+----------+------------+------------------------------------------------+
7 rows in set, 1 warning (0.00 sec)
```

上述操作中，MySQL 在执行 LIMIT 操作时是从第 1 页(偏离量减 1)获取到指定的页数(如 8999 页)，然后再舍弃前面的页中数据，保留指定页数中的数据。因此，当获取的页码越大时，LIMIT 后的偏移量值越大，而偏移量值越大，就会导致查询所花费的时间越长。

所以，针对以上的情况，在实际开发中经常会在业务上限制获取数据的页数，如限制获取的页数不能超过 40 页。除此之外，也可以修改查询的语句，使用 WHERE 判断查询的偏移量，让 LIMIT 仅限制获取的数据量。例如，将以上的 SQL 语句修改成以下形式，具体如下。

```
SELECT * FROM mydb.my_user WHERE id > 90000 ORDER BY id LIMIT 10;
SELECT * FROM mydb.my_user WHERE id > 900000 ORDER BY id LIMIT 10;
```

然后再利用 SHOW PROFILES 查询执行的精确时间分别为 0.00139575 秒和 0.00065400 秒，可以明显地看出执行时间加快了很多。需要注意的是，通过修改 SQL 语句的方式优化 LIMIT 分页，排序字段最好从 1 开始，并且能够连续自增，这样才能方便获取数据的信息。因此，在实际中很少使用这种方式，通常会利用在业务上限制页数的方式优化 LIMIT 分页。

6. 查询缓存

对于数据库优化来说，要想使每次用户请求的数据都能够快速响应，对于数据更新不频繁、数据结构不经常变化的情况可以考虑采用查询缓存。查询缓存指的是 MySQL 服务器提供的一种对 SELECT 语句查询结果的缓存机制。下面通过以下语句查看当前 MySQL 中缓存的相关设置。

```
mysql> SHOW VARIABLES  LIKE '%query_cache%';
+------------------------------+----------+
| Variable_name                |  Value   |
+------------------------------+----------+
| have_query_cache             | YES      |
| query_cache_limit            | 1048576  |
| query_cache_min_res_unit     | 4096     |
| query_cache_size             | 1048576  |
| query_cache_type             | OFF      |
```

```
| query_cache_wlock_invalidate          | OFF       |
+-------------------------------+----------+
6 rows in set, 1 warning (0.01 sec)
```

在上述结果中，query_cache_size 用于设置查询缓存的内存大小，默认为 1048576 字节(1MB)，query_cache_type 用于开启查询缓存功能，0(OFF)表示关闭，1(ON)表示开启，2(DEMAN)表示只有 SELECT 选项设置为 SQL_CACHE 才会缓存查询结果，其他变量的含义如表 11-9 所示。

表 11-9　与缓存相关的变量

变量名称	描　　述
have_query_cache	YES 表示当前 MySQL 版本支持查询缓存
query_cache_limit	单个结果集所被允许缓存的最大值，默认为 1MB
query_cache_min_res_unit	每个被缓存的结果集要占用的最小内存，默认为 4KB
query_cache_wlock_invalidate	当写锁加在表上时，是否先让该表相关的查询缓存失效，1 表示失效，0 表示有效

下面以缓存 pid 为 3 且编号最靠前的两位员工信息为例进行演示，具体步骤如下。

(1) 开启查询缓存。打开 MySQL 的配置文件 my.ini，开启查询缓存，设置缓存大小为 64MB。具体如下。

```
query_cache_type=1
query_cache_size=67108864
```

在上述配置中，缓存的大小在设置时需要将 64MB 换算为 67108864 字节。保存后，重启 MySQL 服务器使配置生效。

(2) 对比缓存查询结果。按要求执行查询语句，对比前后两次的查询结果。具体如下。

```
mysql>SELECT id,name FROM mydb.my_user WHERE pid=3 ORDER BY id LIMIT 2;
+----+-------+
| id  | name   |
+----+-------+
|  1  | ybKxg  |
|  5  | oUGwq  |
+----+-------+
2 rows in set (0.03 sec)
mysql>SELECT id,name FROM mydb.my_user WHERE pid=3 ORDER BY id LIMIT 2;
+----+-------+
| id  | name   |
+----+-------+
|  1  | ybKxg  |
|  5  | oUGwq  |
+----+-------+
2 rows in set (0.00 sec)
```

上述操作中，开启查询缓存后，会缓存第 1 次的查询结果，再次使用相同的 SELECT 语

句时，则直接从缓存中返回结果。从以上对比可以看出，第 1 次查询的时间大约为 0.03 秒，第 2 次查询的时间大约为 0 秒，查询速度明显变快。读者也可以开启 profile 机制获取更加精确的时间。

另外，在使用查询缓存时还需要注意以下几点，以免出现与预期不一样的效果。

- 缓存的数据有变化或数据表的结构有变化时，服务器会自动清空全部缓存数据，使缓存失效。
- 查询的语句中获取或使用了动态变化的数据则不会生成缓存，如使用 now()获取时间，rand()获取随机数等。
- select 选项使用了 SQL_NO_CACHE，对查询的数据不进行缓存。其中，select 选项值 SQL_CACHE 和 SQL_NO_CACHE 在 MySQL 5.7.20 开始已不推荐使用，并将在 MySQL 8.0 中删除，请读者慎重使用。
- 相同的查询语句，会因语句中多一个或少一个空格，以及大小写问题而分别生成多个缓存，从而造成缓存空间的浪费。

(3) 查看缓存使用情况。在开启查询缓存后，可以利用 MySQL 提供的 SHOW STATUS 监视查询缓存的性能，具体 SQL 语句及执行结果如下。

```
mysql> SHOW STATUS LIKE "%Qcache%";
+-------------------------+----------+
| Variable_name           | Value    |
+-------------------------+----------+
| Qcache_free_blocks      | 1        |
| Qcache_free_memory      | 67096608 |
| Qcache_hits             | 3        |
| Qcache_inserts          | 3        |
| Qcache_lowmem_prunes    | 0        |
| Qcache_not_cached       | 0        |
| Qcache_queries_in_cache | 3        |
| Qcache_total_blocks     | 8        |
+-------------------------+----------+
8 rows in set (0.01 sec)
```

在上述查询结果中，Qchache_free_memory 表示剩余的查询缓存空间大小，Qcache_hits 表示缓存的命中次数，Qcache_inserts 表示缓存的插入次数，Qcache_queries_in_cache 表示当前缓存的数量。

至此，一个常见的数据库优化已全部完成，在开发中读者还需根据实际的需求均衡、性能的提升和系统资源开销等情况后再确定数据库的优化方案。

11.8　本章小结

本章主要讲解了与数据库优化密切相关的常见存储引擎的特性，索引的应用以及使用原则，锁机制的特点及相关操作。另外还包括数据库优化的常见操作，如分表分区技术、数据碎片的整理、慢查询日志、查询缓存、LIMIT 分页优化等。希望通过本章的学习，读者能够掌握数据库优化相关的理论与实际操作，具备解决实际问题的能力。

11.9 课后练习

一、填空题

1. MySQL 5.7 中默认的存储引擎是________。
2. 若索引是从左开始截取数据表中字段值的一部分内容，这种索引可以统称为________。
3. InnoDB 表在添加行级锁前，服务器必须添加________。
4. MySQL 中分区技术不适用于________、CSV 和 FEDERATED 存储引擎。
5. ________语句可以整理 MyISAM 表中的数据碎片。

二、判断题

1. 一张 InnoDB 数据表有且仅能有一个聚簇索引。(　　)
2. 同一张表的所有分区使用的存储引擎可以不同。(　　)
3. 系统变量 query_cache_type 用于设置是否开启查询缓存。(　　)
4. MyISAM 表比 InnoDB 表数据写入速度快。(　　)
5. LOCK TABLE 锁定的数据表会在会话结束后自动释放。(　　)

三、选择题

1. next-key 锁是由记录锁和(　　)组成的。
   A. 表锁　　B. 意向锁
   C. 间隙锁　　D. 以上选项都不正确
2. 下面关于索引的说法正确的是(　　)。
   A. 重复值较高的字段适合创建普通索引
   B. 数据更新频繁的字段适合创建索引
   C. 存储空间较小的字段适合创建索引
   D. 以上说法全部正确
3. 下列选项中，(　　)可实现多版本并发控制功能。
   A. InnoDB　　B. MyISAM　　C. MEMORY　　D. CSV
4. 下面关于分区的描述错误的是(　　)。
   A. 一张数据表最多仅可以创建 1024 个分区
   B. 分区名称不区分大小写
   C. 子分区算法可以是 LIST、RANGE、HASH 和 KEY 中的任意一种
   D. 以上说法都不正确
5. 以下选项中，(　　)用于设置单个查询缓存结果集占用的最小内存。
   A. query_cache_limit　　B. query_cache_type
   C. query_cache_size　　D. query_cache_min_res_unit

四、简答题

1. 请说出 MyISAM 与 InnoDB 存储引擎至少 5 条区别。
2. 请简述 profile 机制的意义以及使用场景。

五、实训题

1. 为 shop.sh_order_goods 数据表在 order_id 和 goods_id 字段上创建一个唯一性的复合索引。

2. 创建一个与 shop.sh_user 表相同结构的数据表 mydb.p_user，使用 HASH 算法将数据分到 5 个区中存储。

第12章 数据库配置与部署

学习目标

- 掌握 Linux 环境下 MySQL 的安装和配置
- 掌握 MySQL 的数据备份与还原
- 掌握 MySQL 的主从复制、读写分离配置

Linux 是一种源代码开放和自由传播的计算机操作系统，在服务器领域的应用非常普遍。MySQL 作为一个开源免费的数据库产品，被大量应用在 Linux 平台中。本章将针对 Linux 环境下 MySQL 的安装、使用、配置和部署进行详细讲解。

12.1 Linux 环境安装 MySQL

MySQL 具有良好的跨平台性，它可以很好地在 Linux 环境中使用。考虑到 Linux 有大量的发行版本，不同的版本存在一定的区别，为了使学习更加顺利，本节将从 Linux 环境搭建开始，详细介绍 Linux 环境下 MySQL 安装和配置的完整流程。

12.1.1 Linux 环境搭建

目前，Linux 有许多发行版本，其中 Ubuntu、Fedora 等比较适合个人计算机使用，而 Red Hat Enterprise Linux、CentOS 比较适合服务器使用。对于初次接触 Linux 的新手来说，开源免费的 CentOS 是最好的选择，所以本书将以 CentOS 7.5 版本为例，讲解 Linux 环境下 MySQL 的安装。

1. 准备工作

读者需要到网上下载 CentOS 7.5 的迷你版镜像 CentOS-7-x86_64-Minimal-1804.iso。为了方便在 Windows 操作系统中使用 CentOS，读者还需要安装虚拟机软件，本书选择以 VMware 虚拟机为例进行讲解。下面通过图 12-1 演示虚拟机和物理机的关系。

在图 12-1 中，虚拟机是通过 VMware 软件模拟出的具有硬件系统的功能、运行在一个隔离环境中的计算机系统。在虚拟机中安装 CentOS 后，就可以实现在物理机 Windows 操作系统中使用 CentOS 操作系统。

2. 创建虚拟机

在 VMware 中执行【文件】→【新建虚拟机】，根据新建虚拟机向导操作即可。其中，“安

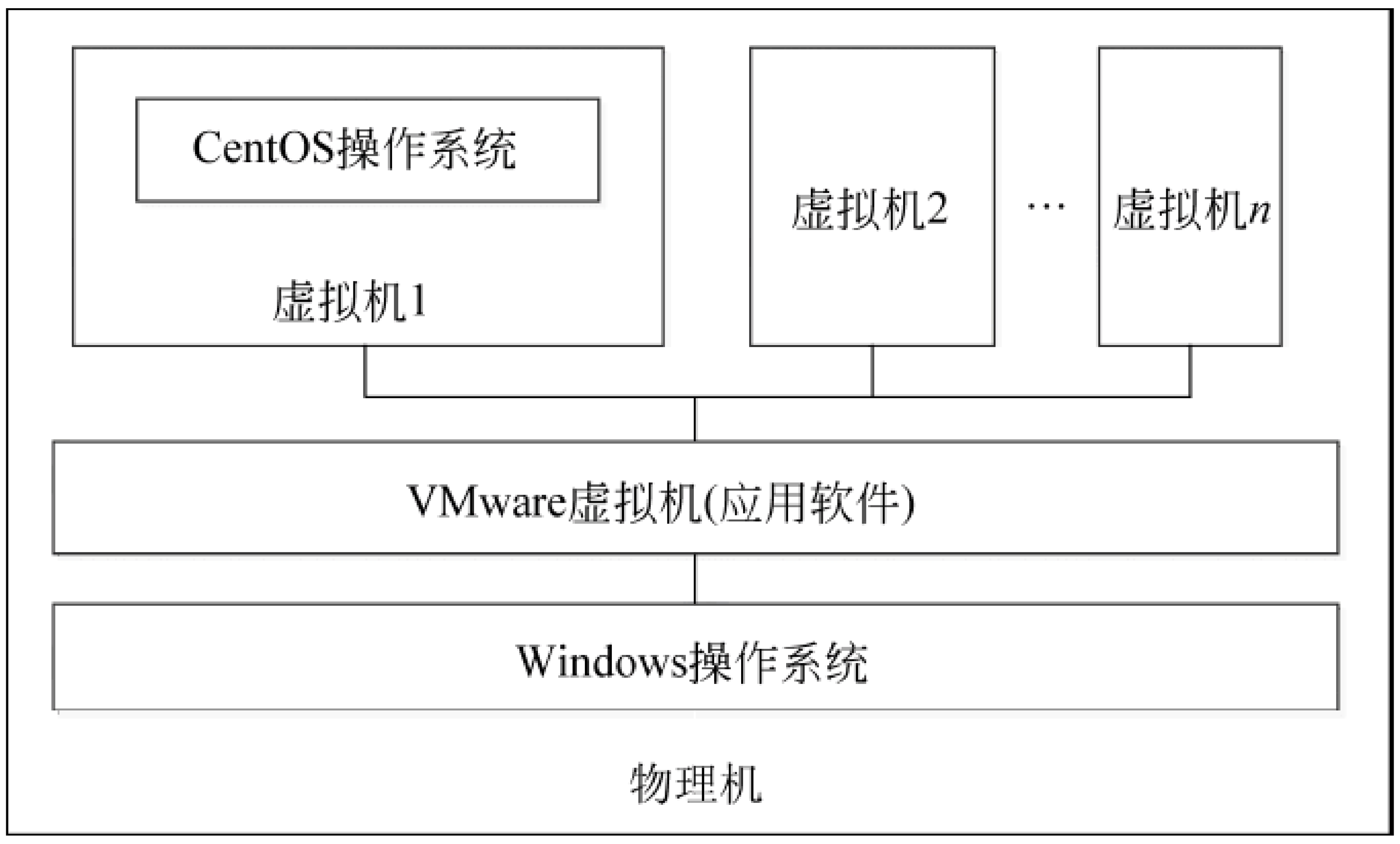


图 12-1　虚拟机和物理机

装程序光盘映像文件(iso)”选择 CentOS-7-x86_64-Minimal-1804.iso 文件，VMware 会提示“已检测到 CentOS7 64 位”。在新建虚拟机时需要对虚拟机的硬件进行配置，下面通过表 12-1 列举具体硬件参数。

表 12-1　虚拟机的硬件参数

硬　　件	说　　明
处理器	根据物理机的核心数设置虚拟机的总核心数，数量越多速度越快
内存	根据物理机的可用内存来设定，至少 1GB
硬盘	至少需要 12GB 的存储空间
CD/DVD	选择 ISO 镜像文件 CentOS-7-x86_64-Minimal-1804.iso
网络适配器	使用 NAT 模式
USB 控制器	不需要此虚拟设备，可移除
声卡	不需要此虚拟设备，可移除
打印机	不需要此虚拟设备，可移除
显示器	使用默认值即可

3. 安装 CentOS

创建虚拟机后，开始安装 CentOS 操作系统。刚开机时，使用鼠标单击虚拟机画面，会出现一个开机菜单，如图 12-2 所示。

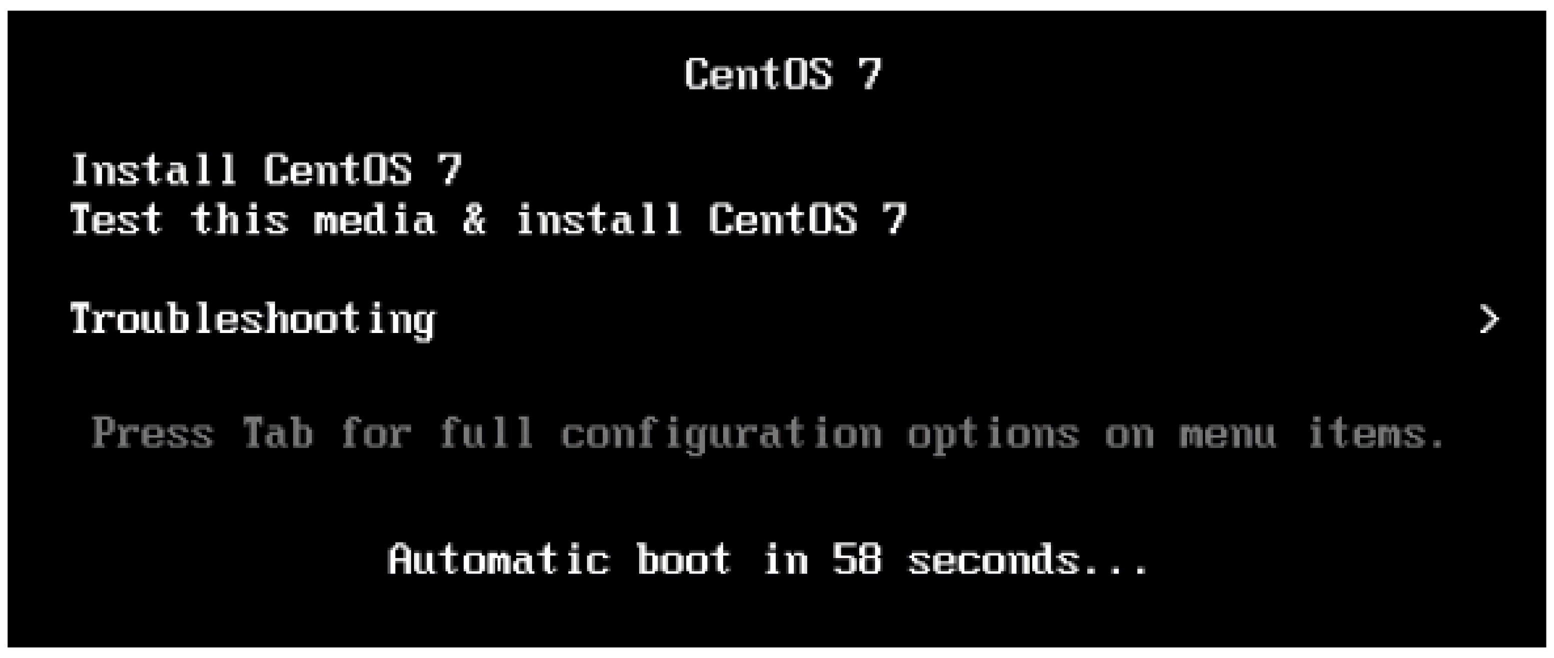


图 12-2　开机菜单

按键盘的↑方向键，选择第一项“Install CentOS 7”，然后等待欢迎界面出现。在欢迎界面中，会提示用户选择语言，选择简体中文即可。然后会出现“安装信息摘要”界面，此时“开始安装”按钮是禁用状态，需要按照如下步骤进行操作。

(1) 配置安装位置。单击“安装位置”按钮，进入“安装目标位置”界面，无须其他操作，直接单击“完成”按钮，返回“安装信息摘要”界面。

(2) 配置网络和主机名。单击“网络和主机名”按钮，会显示当前已经安装的网卡“以太网(ens32)”，将网卡打开，就会看到 IP 地址、子网掩码等信息，如图 12-3 所示。



图 12-3　打开网卡

虚拟机的网络连接是通过 VMware 虚拟网络功能来实现的，其 IP 地址 192.168.78.128 是 VMware 的 DHCP(动态主机配置协议)服务分配的动态 IP 地址。由于不同计算机的网络环境不同，读者实际获取到的 IP 地址也会不同。若要搭建 MySQL 服务器，则推荐为服务器配置一个静态的 IP 地址，单击“配置”按钮，参考图 12-4 进行配置即可。



图 12-4　配置静态 IP 地址

小提示：

（1）读者应根据自身网络环境配置静态 IP，而不要照搬图 12-4 中填写的 IP。

（2）若要更改 VMware 虚拟网络配置，可以执行【编辑】→【虚拟网络配置】，选择 NAT 模式 VMnet8 网卡，然后就可以更改子网 IP、子网掩码，网关 IP、DHCP 地址池。

（3）开始安装。将网卡配置完成后，单击“完成”按钮返回“安装信息摘要”界面。然后单击“开始安装”按钮开始安装。在安装过程中，需要设置 root 用户的密码，为了方便使用，将密码设置为 123456。

（4）重启。当 CentOS 安装成功后，单击“重启”按钮，然后等待 CentOS 系统启动。启动成功后，会看到如图 12-5 所示的登录界面。

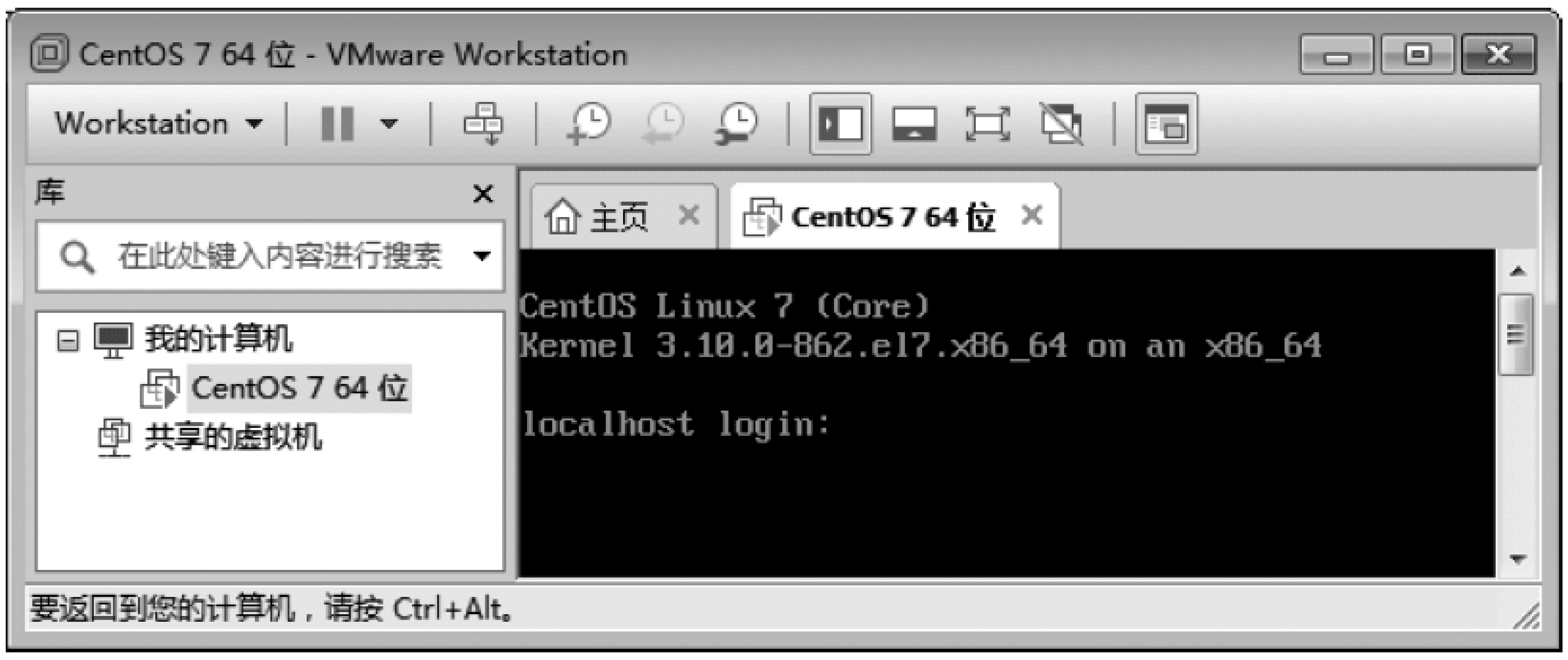


图 12-5　CentOS 登录界面

使用鼠标单击虚拟机画面，输入用户名 root 按回车键，再输入密码 123456 按回车键，即可登录到 CentOS 系统中。

小提示：

（1）VMware 的 NAT 网络模式会在物理机中创建一个虚拟网卡，其名称为 VMware Virtual Ethernet Adapter for VMnet8（简称 VMnet8），用于将物理机和虚拟机连接到同一个子网中，使物理机和虚拟机可以互相访问。但在实际操作时，应注意物理机和虚拟机的防火墙可能会阻挡访问请求。本书后面会介绍防火墙的配置。

（2）由于在 VMware 中操作 CentOS 不太方便，读者可以通过一些远程终端软件来操作，如 Xshell、WinSCP 等，可以方便地进行命令的复制、粘贴。在连接时，选择 SSH 协议、22 端口号，输入 IP 地址（192.168.78.128）、用户名（root）和密码（123456）即可。

4. 创建快照

由于在后面的小节中会讲到两种不同的 MySQL 安装方式，为了避免冲突，建议使用 VMware 为当前虚拟机拍摄一个快照，执行【虚拟机】→【快照】→【拍摄快照】即可。通过快照可以方便地进行系统备份还原，如果在安装 MySQL 过程中发生错误，或者需要换成另外一种方式安装 MySQL 时，利用快照可以将虚拟机还原到拍摄时的状态。

12.1.2　用 yum 安装 MySQL

YUM（Yellow dog Updater, Modified）是 Red Hat、CentOS、Fedora 系统中提供的软件包管理器，它可以从指定的服务器下载软件包，并自动处理依赖关系。

由于目前 CentOS 提供的 YUM 源中没有 MySQL，因此需要在 MySQL 官方网站中获取 MySQL 的 YUM 仓库，然后再安装 MySQL，具体步骤如下。

1. 获取 MySQL 的 YUM 仓库

在 MySQL 官方网站找到 MySQL 的 YUM 仓库下载地址，如图 12-6 所示。

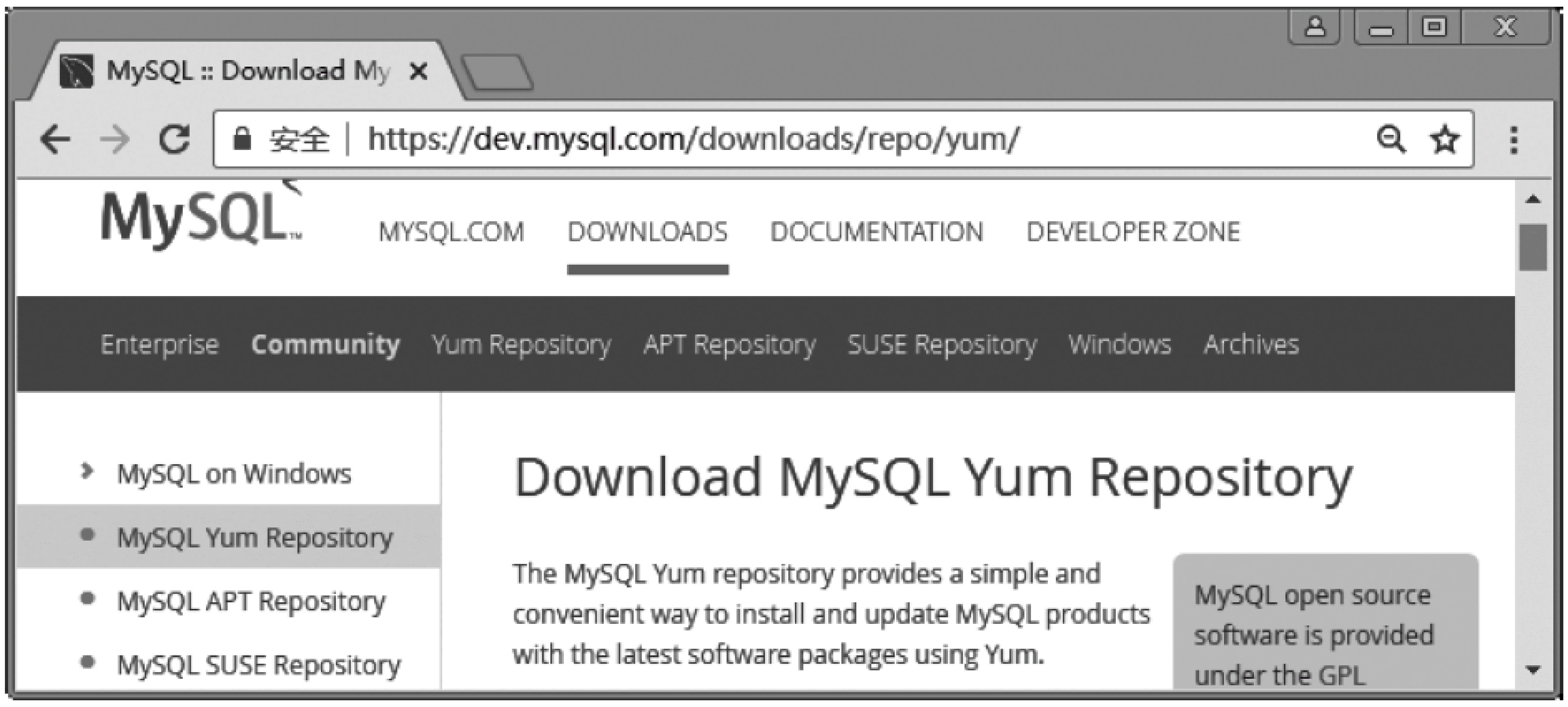


图 12-6 下载 MySQL 的 YUM 仓库

在图 12-6 所示的网页中找到 Red Hat Enterprise Linux 7 的 RPM 包下载。通过下载页面提供的下载链接可以获取下载地址，参考地址如下。

```
https://dev.mysql.com/get/mysql80-community-release-el7-1.noarch.rpm
```

由于 RPM 包需要在 CentOS 中使用，接下来通过 yum 命令安装 wget 文件下载工具，并使用 wget 来下载 RPM 包，具体命令如下。

```
#① 安装 wget
[localhost@root ~]#yum -y install wget
#② 使用 wget 下载 RPM 包
[localhost@root ~]#wget https://dev.mysql.com/get/mysql80-community-release-el7-1.noarch.rpm
```

在上述命令中，yum 的选项-y 表示自动同意询问(方便操作)，install 表示安装指定的包，wget 是包名。wget 用于下载指定地址的文件，下载后会保存在当前目录。

将 RPM 包下载后，执行“rpm -Uvh 文件名”安装 RPM 包，具体命令如下。

```
[localhost@root ~]#rpm -Uvh mysql80-community-release-el7-1.noarch.rpm
```

上述命令中，选项 U 表示升级软件包，v 表示显示详细信息，h 表示显示安装进度。

2. 切换 MySQL 版本

安装 MySQL 的 YUM 仓库后，通过“yum repolist all | grep mysql”命令查看 YUM 中可用的 MySQL 版本，执行结果如下。

```
[root@localhost ~]# yum repolist all | grep mysql
⋮(此处省略一些信息)
mysql57-community/x86_64  MySQL 5.7 Community Server          禁用
mysql57-community-source  MySQL 5.7 Community Server - Sour   禁用
mysql80-community/x86_64  MySQL 8.0 Community Server          启用: 17
mysql80-community-source  MySQL 8.0 Community Server - Sour   禁用
```

从上述结果可知，当前启用的 MySQL 版本为 8.0。为了安装 MySQL 5.7 版本，需要执行如下命令，将版本切换到 5.7。

```
#① 安装 yum-utils(提供了 yum-config-manager 命令)
[root@localhost ~]# yum -y install yum-utils
#② 禁用 MySQL 8.0
[root@localhost ~]# yum-config-manager --disable mysql80-community
#③ 启用 MySQL 5.7
[root@localhost ~]# yum-config-manager --enable mysql57-community
```

完成上述操作后，再次查看 MySQL 版本，会发现已经切换到 5.7 版本。

```
[root@localhost ~]# yum repolist all | grep mysql
⋮(此处省略一些信息)
mysql57-community/x86_64  MySQL 5.7 Community Server          启用:267
mysql57-community-source  MySQL 5.7 Community Server - Sour   禁用
mysql80-community/x86_64  MySQL 8.0 Community Server          禁用
mysql80-community-source  MySQL 8.0 Community Server - Sour   禁用
```

小提示：由于 MySQL 的 YUM 仓库会一直更新，读者可能会获取到更新的版本，如果安装方式与这里介绍的不一致，可以阅读下载页面中的帮助文档进行学习。

3．安装 MySQL

使用 yum 命令安装 MySQL 非常简单，执行如下命令即可安装。

```
[root@localhost ~]# yum -y install mysql-community-server
```

在将 MySQL 安装完成后，其关键文件的保存路径如表 12-2 所示。

表 12-2　MySQL 关键文件保存路径

文　　件	保 存 路 径
客户端程序	/usr/bin/mysql
服务器程序	/usr/sbin/mysqld
配置文件	/etc/my.cnf
数据目录	/var/lib/mysql
错误日志	/var/log/mysqld.log
字符集、语言等	/usr/share/mysql

4. 启动 MySQL 服务

将 MySQL 安装完成后，执行如下命令即可启动 MySQL 服务。

```
[root@localhost ~]#systemctl start mysqld.service
```

执行如下命令，查看 mysqld 是否监听 3306 端口。

```
[root@localhost ~]#ss -tnlp | grep mysql
LISTEN    0    80    :::3306    :::*    users:(("mysqld",pid=2709,fd=21))
```

看到上述信息，说明 MySQL 已经启动成功，并监听了 3306 端口。

小提示：

（1）使用 yum 命令安装 MySQL 后，会开机自动启动 MySQL 服务。

（2）使用 systemctl restart mysqld 命令可以重启 MySQL 服务，使用 systemctl stop mysqld 命令可以停止 MySQL 服务。

5. 查看 MySQL 中 root 用户的密码

MySQL 服务在首次启动时，会自动初始化数据库，为 root 用户自动设置一个随机的复杂密码，保存在错误日志中。查看 root 用户的密码，具体命令如下。

```
[root@localhost ~]#grep 'temporary password' /var/log/mysqld.log
2018-07-17T09:02:06.265505Z 1 [Note] A temporary password is generated
for root@localhost: 3kBewe+mry1,
```

从上述信息可知，root 用户的密码为“3kBewe+mry1,”（包括最后的逗号）。由于密码是随机生成的，读者在操作时应查看自己系统中生成的密码。

接下来使用这个随机密码登录到 MySQL，具体命令如下。

```
[root@localhost ~]#mysql -uroot -p3kBewe+mry1,
```

上述命令将密码写在了选项-p 后面。如果不希望显示密码，可以省略-p 后面的密码，命令执行后，MySQL 会提示用户输入密码，在输入时不会回显。

上述命令执行后，即可登录成功。由于随机密码比较复杂，为了方便使用，可以执行如下操作，允许使用弱密码，并将密码设置为 123456。

```
#① 设置全局环境变量,允许使用弱密码 (默认为 1)
mysql>SET GLOBAL validate_password_policy=0;
#② 设置全局环境变量,允许使用 4 位密码 (默认为 8)
mysql>SET global validate_password_length=4;
#③ 修改 root 用户密码为弱密码 123456
mysql>ALTER USER 'root'@'localhost' IDENTIFIED BY '123456';
```

小提示：在使用 yum 方式安装 MySQL 后，若需要学习编译安装 MySQL，可以利用 VMware 将当前状态保存为快照，然后还原快照到未安装 MySQL 的状态，再来进行安装。

12.1.3 编译安装 MySQL

编译安装是指下载 MySQL 的源代码，然后使用编译工具(cmake、gcc-c++ 等)将源代码编译成二进制文件，再进行安装。编译安装的优势在于灵活性更强，用户可以修改 MySQL 的源代码后重新编译，或者在编译时指定详细的参数。

1. 获取 MySQL 的源代码

在 MySQL 官方网站找到 MySQL 的源代码下载地址，如图 12-7 所示。

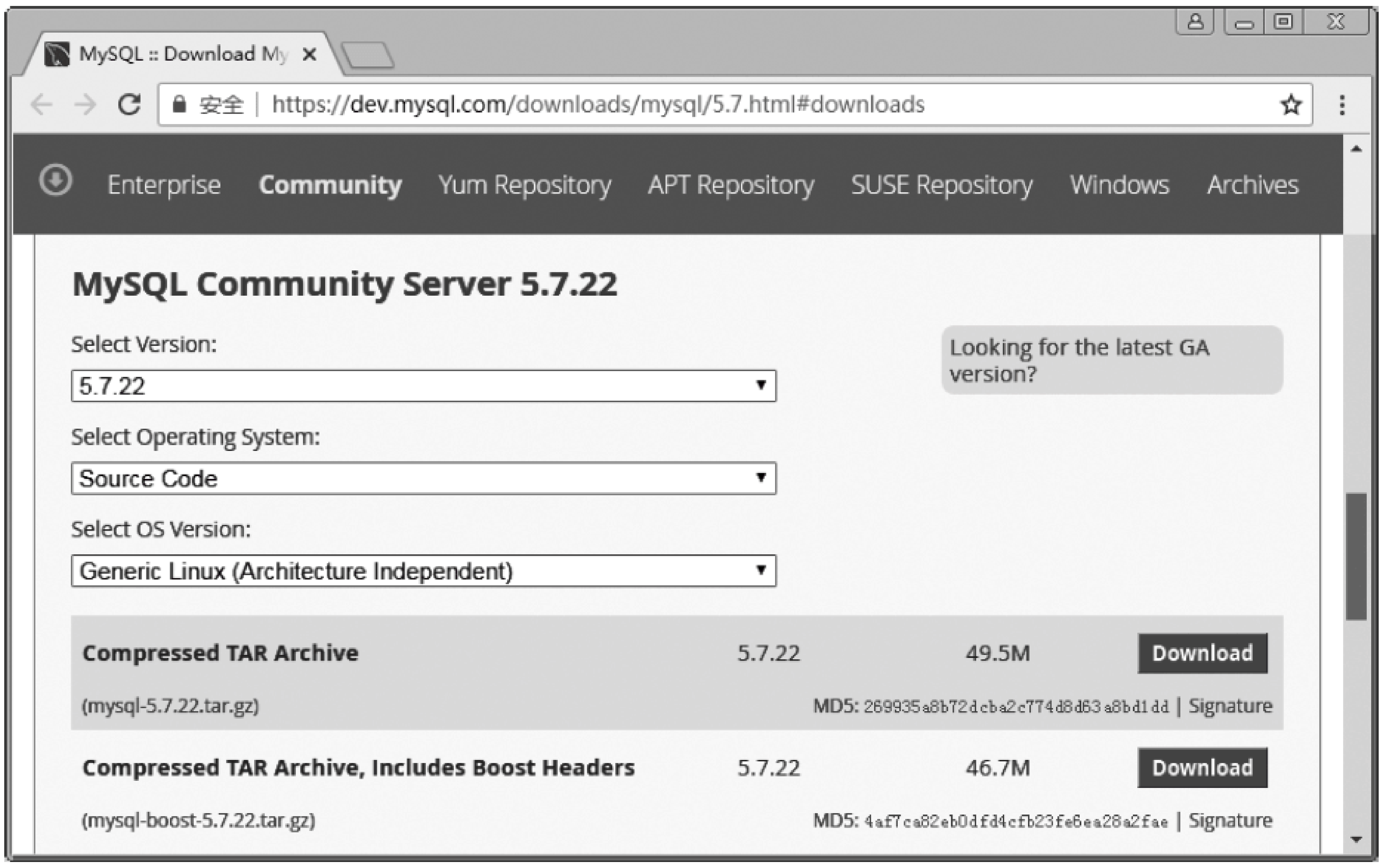


图 12-7 下载 MySQL 源代码

在图 12-7 中，MySQL 提供了两种版本的源代码，文件名分别为 mysql-5.7.22.tar.gz 和 mysql-boost-5.7.22.tar.gz，其区别是第 2 种版本包含了 Boost 头文件，而第 1 种不包含。为了方便安装，这里选择第 2 种版本，即 mysql-boost-5.7.22.tar.gz。获取下载链接后，使用如下命令下载到虚拟机中。

```
#① 安装 wget
[localhost@root ~]# yum -y install wget
#② 使用 wget 下载 MySQL 源码包
[localhost@root ~]# wget https://cdn.mysql.com/Downloads/MySQL-5.7/mysql-boost-5.7.22.tar.gz
```

将 MySQL 源码包下载后，使用“tar -zxvf 文件名”命令将压缩包解压。

```
#① 解压文件
[localhost@root ~]# tar -zxvf mysql-boost-5.7.22.tar.gz
#② 查看解压后的目录
[root@localhost ~]# ls
anaconda-ks.cfg  mysql-5.7.22  mysql-boost-5.7.22.tar.gz
```

2. 安装编译工具和依赖包

MySQL 是使用 C 和 C++ 语言编写的，为了编译 MySQL，需要安装 gcc-c++ 和 cmake 工具。且 MySQL 依赖于 ncurses 字符终端处理库，需要先安装 ncurses-devel 依赖包才能够正确编译。使用 yum 命令下载这些工具和包即可，具体命令如下。

```
[root@localhost ~]# yum -y install gcc-c++ cmake ncurses-devel
```

3. 编译 MySQL

编译 MySQL 共分为两步，第 1 步是执行 cmake 生成 Makefile 文件，第 2 步是执行 make 命令编译。在使用 cmake 时需要指定编译选项，具体可以通过“ccmake .”命令进行查看或编辑，下面通过表 12-3 列举一些常用的编译选项。

表 12-3　MySQL 5.7 常用编译选项

选　项　名	说　明	默　认　值
CMAKE_INSTALL_PREFIX	安装路径	/usr/local/mysql
MYSQL_DATADIR	数据目录	/uar/local/mysql/data
DOWNLOAD_BOOST	下载 boost	OFF
WITH_BOOST	指定 boost 路径	(空)
WITH_EXTRA_CHARSETS	安装额外字符集	all
WITH_INNOBASE_STORAGE_ENGINE	InnoDB 存储引擎	ON
WITH_SYSTEMD	支持 systemd	OFF

一般情况下，这些编译选项使用默认值即可。接下来开始编译安装 MySQL，在编译前，先使用 cd 命令切换到 MySQL 源代码目录中。

```
[root@localhost ~]# cd mysql-5.7.22
```

然后执行如下命令，编译安装 MySQL。

```
#① 切换到 mysql-5.7.22 目录中
[root@localhost mysql-5.7.22]# cmake -DWITH_BOOST=boost -DWITH_SYSTEMD=1
#② 执行 make
[root@localhost mysql-5.7.22]# make
```

在上述命令中，cmake 的选项“-D”用于指定编译选项，紧跟其后的 WITH_BOOST 是选项名，其值 boost 表示使用当前目录下的 boost，即“/root/mysql-5.7.22/boost”。由于 CentOS 7 支持 systemd，因此在编译选项中增加了对 systemd 的支持。

小提示：

（1）如果下载的 MySQL 源码包文件名为 mysql-5.7.22.tar.gz，该源码包中不包含 boost，需要在编译选项中加上“-DDOWNLOAD_BOOST=1”下载 boost。

（2）添加“-DWITH_SYSTEMD=1”编译选项后，会在源码包的 scripts 目录中生成 mysqld.service 文件，该文件用于 systemctl 管理 MySQL 服务(在后面会用到)。

（3）make 命令支持并行编译，以加快编译速度。开启并行编译的命令为“make -j 数字”，数字表示并行数。MySQL 并行编译非常占用内存，经过测试，将数字设为 3 时需要虚拟机具备 3G 的内存，如果内存不足会导致编译失败。

4．安装 MySQL

将 MySQL 编译完成后，就可以进行安装，具体命令如下。

```
[root@localhost mysql-5.7.22]#make install
```

安装成功后，切换到/usr/local/mysql 目录查看文件是否存在，如下所示。

```
#① 切换目录
[root@localhost mysql-5.7.22]#cd /usr/local/mysql
#② 查看文件列表
[root@localhost mysql]#ls
bin  COPYING  COPYING-test  docs  include  lib  man  mysql-test
README  README-test  share  support-files
```

从上述结果可以看出，MySQL 已经安装完成。

5．创建配置文件

将 MySQL 安装完成后，还需要创建配置文件，具体操作如下。

```
#① 先备份 CentOS 自带的配置文件(重命名 my.cnf 为 my.cnf.bak)
[root@localhost mysql]#mv /etc/my.cnf /etc/my.cnf.bak
#② 创建一个新的配置文件
[root@localhost mysql]#vi /etc/my.cnf
```

上述命令执行后，会进入到 vi 编辑器，用于输入文本内容。在 vi 编辑器模式下，按 i 键进入到编辑模式，会看到屏幕左下角出现“-- INSERT --”，此时就可以输入如下内容。

```
[mysqld]
datadir=/var/lib/mysql
socket=/tmp/mysql.sock
port=3306
log-error=/var/lib/mysql/mysqld.log
pid-file=/var/lib/mysql/mysqld.pid
symbolic-links=0
user=mysql
```

输入完成后，按 ESC 键退出编辑模式，然后输入“:wq”命令并按回车键，即可保存并退出 vi 编辑器。在上述配置中，datadir 表示数据目录，socket 表示 sock 文件路径(用于 socket 连接)，port 表示端口号，log-error 表示错误日志路径，pid-file 表示进程 id 保存路径，symbolic-links 设为 0 表示禁用符号链接，user 表示 MySQL 的工作用户。

在 Linux 系统中，禁用符号链接，为 MySQL 指定一个专门的用户，都是为了提高安全性，降低被黑客攻击的风险。下面还需要创建 mysql 用户，具体操作如下。

```
#创建 mysql 用户组和用户，并禁止登录
[root@localhost mysql]#groupadd mysql
[root@localhost mysql]#useradd -r -M -g mysql -s /bin/false mysql
```

在上述命令中，groupadd 用于创建用户组，useradd 用于创建用户，用户名和组名都是 mysql。useradd 的选项-r 表示创建系统用户，-M 表示不创建用户目录，-g mysql 表示加入 mysql 用户组，-s /bin/false 表示禁止登录。

6. 初始化数据库

将配置文件准备好后，就可以开始初始化数据库，具体操作如下。

```
#① 初始化数据库，并忽略安全性(MySQL 中的 root 用户初始密码为空)
[root@localhost mysql]#./bin/mysqld --initialize-insecure
#② 切换到数据目录
[root@localhost mysql]#cd /var/lib/mysql
#③ 查看数据目录中的文件
[root@localhost mysql]#ls
auto.cnf        ibdata1       ib_logfile1  mysqld.log          sys
ib_buffer_pool  ib_logfile0   mysql        performance_schema
```

从上述结果可以看出，MySQL 初始化数据库后，就会在数据目录中自动生成一些文件，这些文件包括数据库中的数据、日志等。

7. 管理 MySQL 服务

将 MySQL 提供的 mysqld.service 服务脚本复制到 CentOS 的 systemd 目录中，就可以使用 systemctl 命令管理 MySQL 服务，具体操作如下。

```
#① 复制 mysqld.service 脚本
[root@localhost mysql]#cp /root/mysql-5.7.22/scripts/mysqld.service /usr/lib/systemd/system/
#② 启动 MySQL 服务
[root@localhost mysql]#systemctl start mysqld.service
#③ 查看 MySQL 服务是否启动
[root@localhost mysql]#ss -tnlp | grep mysql
LISTEN   0     80    :::3306    :::*     users:(("mysqld",pid=2709,fd=21))
#④ 设置 MySQL 服务的开机启动
[root@localhost mysql]#systemctl enable mysqld
#⑤ 查看 MySQL 服务是否已经允许开机启动
[root@localhost mysql]#systemctl list-unit-files | grep mysqld
mysqld.service            enabled
```

8. 登录 MySQL

在 MySQL 服务启动成功后，使用客户端工具登录 MySQL，具体操作如下。

```
#① 切换到客户端工具所在的目录
[root@localhost mysql]# cd /usr/local/mysql/bin
#② 运行客户端程序,登录 MySQL
[root@localhost bin]# ./mysql -uroot
#③ 设置密码为 123456
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY '123456';
mysql> exit
```

由于在使用客户端工具时,需要先切换目录,比较麻烦。可以在/usr/local/bin 目录中创建一个软链接,实现在任意目录中直接通过 mysql 命令启动客户端,具体操作如下。

```
#① 创建软链接
[root@localhost bin]# ln -s `pwd`/mysql /usr/local/bin/mysql
#② 切换到 root 用户的家目录
[root@localhost bin]# cd
#③ 登录 MySQL 测试
[root@localhost ~]# mysql -uroot -p123456
```

在上述命令中,ln 的选项-s 表示创建软连接,第 1 个参数是源文件地址(`pwd`用于获取当前目录的路径),第 2 个参数是目标地址。在创建软链接后,就可以使用 mysql 命令方便地启动 MySQL 客户端工具了。

多学一招:创建远程登录用户

对于独立的 MySQL 服务器,客户端和服务器不在同一台电脑上,为了允许远程客户端访问数据库,就需要在 MySQL 中创建远程登录用户。

为了方便测试,下面在 CentOS 环境的 MySQL 服务器中创建一个远程用户 mydb,并赋予同名数据库 mydb 的操作权限,具体操作如下。

```
#① 创建 mydb 数据库
mysql> CREATE DATABASE mydb;
#② 创建 mydb 用户,密码为 123456,远程地址为 192.168.78.1
mysql> CREATE USER 'mydb'@'192.168.78.1' IDENTIFIED BY '123456';
#③ 为 mydb 用户分配 mydb 数据库的操作权限
mysql> GRANT ALL PRIVILEGES ON mydb.* TO 'mydb'@'192.168.78.1';
mysql> exit
```

在上述操作中,mydb 用户的远程地址 192.168.78.1 是物理机 Windows 操作系统的 IP 地址。在后面将会通过 Windows 系统中的 MySQL 客户端来登录服务器。

由于 CentOS 的防火墙是打开的,需要在防火墙中开放 3306 端口,具体操作如下。

```
#① 开放 3306 端口
[root@localhost ~]# firewall-cmd --zone=public --add-port=3306/tcp -permanent
#② 执行 reload 使防火墙配置生效
[root@localhost ~]# systemctl reload firewalld.service
```

```
#③ 测试 3306 端口是否已经打开
[root@localhost ~]# firewall-cmd --zone=public --query-port=3306/tcp
yes
```

接下来在Windows系统中打开命令提示符，启动MySQL客户端，登录到MySQL服务器。服务器IP地址为192.168.78.128，用户名为mydb，密码为123456，具体命令如下。

```
C:\mysql5.7\bin>mysql -h192.168.78.128 -umydb -p123456
```

在登录成功后，使用SHOW DATABASES语句查看数据库，结果如下。

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mydb               |
+--------------------+
2 rows in set (0.00 sec)
```

从上述结果可以看出，当前成功登录了MySQL服务器，并可以访问mydb数据库。

12.2 MySQL配置文件

MySQL的配置文件是my.cnf(或my.ini)，利用它可以进行数据库调优、性能优化、主从复制等配置。在前面已经讲过basedir、datadir、port、socket、log-error、pid-file、symbolic-links和user等配置的作用，本节将针对其他常见的配置进行详细讲解。

12.2.1 配置区段

在MySQL配置文件my.cnf中，有多个区段(section)，前面用过的basedir、datadir都写在“[mysqld]”区段中，表示对MySQL服务的配置。my.cnf还可以使用“[client]”区段对客户端进行配置，下面通过示例配置演示这两种区段的区别。

```
[client]
socket=/tmp/mysql.sock          #客户端配置
[mysqld]
socket=/tmp/mysql.sock          #服务器配置
```

上述配置将客户端与服务器的sock文件指向了相同的路径，双方就可以进行socket连接。由于客户端socket默认值为/tmp/mysql.sock，因此可以省略客户端配置。但若将服务端的socket指向其他路径，则客户端也需要配置相同的socket路径。

由于“[client]”区段中的配置并不常用，因此这里不再进行详细讲解，具体可以参考MySQL官方手册。在后面的小节讲解的配置，都是写在“[mysqld]”区段中。

12.2.2 基本配置

在MySQL服务端配置中，有一些常用的基本配置，如SQL模式、默认字符集、默认校对集、默认存储引擎等，下面进行简要说明。

- sql_mode：服务器SQL模式，用于定义MySQL支持的SQL语法以及执行哪种数

据验证检查。除非有特殊需求，一般情况下不需要配置，使用默认值即可。

- character_set_server：服务器的默认字符集，默认为 latin1。
- collation_server：服务器的默认校对集，默认为 latin1_swedish_ci。
- explicit_defaults_for_timestamp：服务器对 TIMESTAMP 列的默认值和 NULL 值的处理方式，默认为 0。设为 1 时，TIMESTAMP 列可以保存 NULL 值。
- max_connections：允许的客户端最大同时连接数，默认为 151。如果连接数过多，会消耗大量系统资源。如果连接数已满，会遇到 Too many connections 错误。
- open_files_limit：操作系统允许 mysqld 打开的文件数，默认为 5000。
- default_storage_engine：默认存储引擎，默认为 InnoDB。

下面以 explicit_defaults_for_timestamp 为例进行演示，该配置既可以在 my.cnf 配置文件中修改，也可以使用 SET 语句临时修改。修改前后的效果对比如下。

```
#① 修改配置前，创建含有时间戳字段的数据表，创建后查看表结构
mysql>CREATE TABLE my_time (t timestamp);
mysql>DESC my_time;
+-------+-----------+------+-----+-------------------+-----------------------------+
| Field | Type      | Null | Key | Default           | Extra                       |
+-------+-----------+------+-----+-------------------+-----------------------------+
| t     | timestamp | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
+-------+-----------+------+-----+-------------------+-----------------------------+
1 row in set (0.01 sec)
#② 修改配置 (默认为 0,改为 1)
mysql>SET explicit_defaults_for_timestamp=1;
#③ 再次创建一个含有时间戳字段的数据表，创建后查看表结构
mysql>CREATE TABLE my_time2 (t timestamp);
mysql>DESC my_time2;
+-------+-----------+------+-----+---------+-------+
| Field | Type      | Null | Key | Default | Extra |
+-------+-----------+------+-----+---------+-------+
| t     | timestamp | YES  |     | NULL    |       |
+-------+-----------+------+-----+---------+-------+
1 row in set (0.00 sec)
```

从上述操作可以看出，更改配置后，新创建的 my_time2 数据表中的时间戳字段不再具有 NOT NULL 约束和 on update CURRENT_TIMESTAMP 属性，且默认值为 NULL。

12.2.3　内存和优化配置

在 MySQL 的配置文件中，还有一些关于服务器的内存和优化配置。通过分配较大的内存空间有利于提高性能，但是也不能设置过高，以免造成操作系统频繁换页，降低系统性能。下面介绍一些常见的关于内存和优化的配置，具体如下。

- key_buffer_size：索引缓冲区大小，由所有线程共享。增加大小可以获得更好处理的索引(对于读取和多次写入)，默认值为 8M。
- table_open_cache：所有线程的打开表的缓存数量，用于更快地访问表内容，默认为 2000。但若设置过大，会占用更多的文件描述符。

- sort_buffer_size：为每个需要进行排序的会话分配的缓冲区大小，默认为 256K。增加该值可以提高 ORDER BY 和 GROUP BY 的速度。
- read_buffer_size：对 MyISAM 表执行顺序扫描的每个线程分配的缓冲区，默认为 128K。如果对表的顺序扫描非常频繁，增加此值可以提高速度。
- thread_cache_size：服务器应缓存多少线程以供重用，默认为－1（自动调整大小）。当客户端断开连接时，如果客户端的线程少于指定值，则将其放入缓存中。
- query_cache_size：表示查询缓冲区的大小，默认为 1M。需要和 query_cache_type 配合使用，当 query_cache_type 的值为 0 时不使用查询缓冲区，为 1 时使用。使用查询缓冲区可以提高查询的速度。
- tmp_table_size：内部临时表的最大值，默认为 16M。如果内存中的临时表超出限制，MySQL 会自动将其转换为磁盘上的临时表。
- innodb_buffer_pool_size：InnoDB 缓存表和索引数据的内存区域，默认为 128M。对于多次访问相同的表数据的情况，配置较大的缓冲池可以减少磁盘 I/O，提高查询速度。
- innodb_log_file_size：InnoDB 重做日志的日志组中每个日志文件的大小，默认为 48M。该值越大，缓冲池中必要的检查点刷新活动就会越少，节省磁盘 I/O。但是日志文件越大，MySQL 崩溃恢复的速度就越慢。
- innodb_lock_wait_timeout：InnoDB 的锁等待超时时间，默认为 50 秒。发生锁定等待超时时，将回滚当前语句（而不是整个事务）。

12.2.4 日志配置

MySQL 中的日志主要分为 4 类，分别是错误日志、常规日志、二进制日志、慢查询日志。关于这些日志的常用配置如下。

- log-error：错误日志的文件路径，保存 MySQL 启动、运行或停止时的日志信息。
- general-log：是否开启常规日志，用于记录客户端连接和执行的 SQL 语句。
- general-log_file：常规日志的文件路径。
- log_slow_queries：慢查询日志文件路径，用于记录超过 long_query_time 时间或没有使用索引的查询。
- long_query_time：当查询时间超过指定的值，就会记录到慢查询日志中。
- log_queries_not_using_indexes：是否在慢查询日志中记录未使用索引的查询。
- log-bin：二进制日志的保存路径，主要用于复制环境和数据恢复。
- max_binlog_size：二进制日志单个文件的大小限制。
- expire_logs_days：自动清除超过指定天数的过期日志。

其中，错误日志和慢查询日志在前面已经用过，二进制日志会在后面详细讲解。下面演示常规日志的使用。首先在“[mysqld]”区段中增加配置，开启常规日志。

```
general-log=1
general-log_file=/var/lib/mysql/general.log
```

更改配置文件后，执行 systemctl restart mysqld 命令使配置生效。然后登录到

MySQL 服务器,随意执行 SQL 语句后退出,就可以在常规日志文件中看到客户端信息和执行过的 SQL 语句,示例结果如下。

```
Tcp port: 3306  Unix socket: /tmp/mysql.sock
Time                           Id Command   Argument
2018-07-20T20:09:16.102691Z    2 Connect    root@localhost on  using Socket
2018-07-20T20:09:16.103588Z    2 Query      select @@version_comment limit 1
2018-07-20T20:09:20.524053Z    2 Query      SHOW DATABASES
2018-07-20T20:09:21.615231Z    2 Quit
```

在上述信息中,Time 表示日志时间,Id 表示连接 id,Command 表示执行的命令,Argument 表示参数。从这些信息可知,连接的用户为 root@localhost,连接后执行了两条 SQL 语句,select @@version_comment limit 1 由 MySQL 客户端自动执行,用于获取版本说明,SHOW DATABASES 是用户执行的 SQL 语句。

12.3　数据备份与还原

在操作数据库时,难免会发生一些意外造成数据丢失。例如,突然停电、设备故障、操作失误等都可能导致数据的丢失。为了确保数据的安全,需要定期对数据库进行备份,当遇到数据库中数据丢失或者出错的情况,就可以将数据进行还原,从而最大限度地降低损失。本节将针对数据的备份和还原进行详细讲解。

12.3.1　数据备份

在 MySQL 的 bin 目录中,提供了 mysqldump 工具,用于将数据库导出成 SQL 脚本,在 SQL 脚本中会包含表结构和数据等信息。执行 SQL 脚本可以导入备份的数据。

mysqldump 工具支持备份单个或多个数据库,下面分别进行讲解。

(1) 备份单个数据库。使用 mysqldump 命令备份单个数据库,语法格式如下。

```
mysqldump -uusername -ppassword dbname [tbname1 [tbname2…]]
```

在上述语法中,-u 后面的 username 表示用户名,-p 后面的 password 表示密码(可以先留空),dbname 表示需要备份的数据库名称,tbname 表示数据库中的表名,可以指定一个或多个表,多个表名之间用空格分隔,如果不指定则备份整个数据库。

(2) 备份多个数据库。使用 mysqldump 命令备份多个数据库,语法格式如下。

```
mysqldump -uusername -ppassword --databases dbname1 [dbname2 …]
```

上述语法格式中,“--databases”参数后面至少应指定一个数据库名称,如果有多个数据库,则名称之间用空格隔开。

(3) 备份所有数据库。使用 mysqldump 命令备份所有数据库,语法格式如下。

```
mysqldump -uusername -ppassword --all-databases
```

上述语法格式中,“--all-databases”选项表示备份所有的数据库。

mysqldump命令会将结果直接输出到屏幕中，为了保存输出结果，需要在命令后面加上“>filename.sql”，表示将输出重定向到filename.sql文件，文件名前面可以加上路径。

需要注意的是，如果备份单个数据库，在生成的SQL脚本中不包含创建数据库和选择数据库的语句，在使用SQL脚本前应先手动创建并选择数据库。而备份多个数据库时，创建的SQL脚本中包含创建和选择数据库的语句。

另外，为了确保数据的一致性，mysqldump在备份每个数据库时会先进行锁表。对于InnoDB存储引擎的数据表，可以通过--single-transaction选项来利用事务避免锁表。

由于mysqldump命令的3种备份方式比较类似，所以下面仅演示备份单个数据库的情况，具体步骤如下。

```
#① 登录 MySQL
[root@localhost ~]#mysql -uroot -p123456
#② 准备一些测试数据
mysql>CREATE DATABASE IF NOT EXISTS mydb;
mysql>CREATE TABLE mydb.goods (id INT, name VARCHAR(32));
mysql>INSERT INTO mydb.goods VALUES (1, 'book'), (2, 'apple');
mysql>exit
#③ 备份数据库
[root@localhost ~]#cd /usr/local/mysql/bin
[root@localhost bin]#./mysqldump -uroot -p123456 mydb >~/mydb.sql
```

上述操作执行成功后，会在root用户目录生成一个mydb.sql备份文件。使用cat命令查看该文件，可以看到如下结果。

```
[root@localhost bin]#cat ~/mydb.sql
--MySQL dump 10.13  Distrib 5.7.22, for Linux (x86_64)
--Host: localhost    Database: mydb
------------------------------------------------------------
--Server version 5.7.22
  ⋮(此处省略一些显示结果)
DROP TABLE IF EXISTS `goods`;
/*!40101 SET @saved_cs_client=@@character_set_client */;
/*!40101 SET character_set_client=utf8 */;
CREATE TABLE `goods` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(32) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client=@saved_cs_client */;
  ⋮(此处省略一些显示结果)
LOCK TABLES `goods` WRITE;
/*!40000 ALTER TABLE `goods` DISABLE KEYS */;
INSERT INTO `goods` VALUES (1,'book'),(2,'apple');
/*!40000 ALTER TABLE `goods` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;
  ⋮(此处省略一些显示结果)
--Dump completed on 2018-07-20 18:00:56
```

从输出结果可以看出，SQL 脚本中会包含 mysqldump 版本、MySQL 服务器版本、主机名、备份的数据库名称，以及一些 SQL 语句。其中以“/ * !”开头、“ * /”结尾的语句都是可执行的 MySQL 注释，这些语句可以被特定版本的 MySQL 执行，但在其他数据库管理系统中将被作为注释忽略，这可以提高数据库的可移植性。

小提示：以“/ * ！ 40101”开头、“ * /”结尾的注释语句中，40101 是 MySQL 的版本号，相当于 MySQL 4.1.1，在还原数据时，如果当前 MySQL 的版本比 MySQL 4.1.1 高，“/ * ！ 40101”和“ * /”之间的内容就被当成 SQL 来执行，如果比当前版本低，被当作注释。

12.3.2　数据还原

在创建备份以后，当数据丢失、损坏，或进行数据库迁移时，就可以利用备份来还原数据。数据还原有两种常用的方式，下面分别进行讲解。

1. 输入重定向

前面介绍过 mysqldump 命令的输出重定向，与之相反，mysql 命令可以使用输入重定向读取 SQL 脚本还原数据，只需在 mysql 命令的末尾加上“<filename. sql”即可。

需要注意的是，若 SQL 脚本文件中不包含创建和选择数据库的语句，则在还原数据前，应先创建数据库，并在 mysql 命令的参数中指定数据库名称，语法示例如下。

```
mysql -uusername -ppassword [dbname] < filename.sql
```

上述语法格式中，username 表示用户名，password 表示密码，dbname 表示数据库名称，指定数据库名称后，相当于执行了“USE 数据库名”语句。filename. sql 是 SQL 脚本文件，文件名前面可以加上路径。

接下来通过案例演示如何利用输入重定向还原数据，具体步骤如下。

```
#① 登录 MySQL
[root@localhost bin]#mysql -uroot -p123456
#② 创建数据库
mysql> CREATE DATABASE mydb2;
mysql> exit
#③ 还原数据
[root@localhost bin]#mysql -uroot -p123456 mydb2 < ~/mydb.sql
#④ 查看数据是否还原
[root@localhost bin]#mysql -uroot -p123456 mydb2
mysql> SELECT id, name FROM goods;
+------+-------+
| id   | name  |
+------+-------+
|   1  | book  |
|   2  | apple |
+------+-------+
2 rows in set (0.00 sec)
```

在上述命令中，“～/mydb. sql”文件是 mysqldump 导出的 SQL 脚本。从查询结果可以看出，数据已经正确还原。

2. source 命令

source 是 MySQL 客户端提供的命令，其语法格式如下。

```
source 文件路径
```

接下来通过案例演示如何利用 source 命令还原数据，具体步骤如下。

```
#① 先删除已有的数据
mysql> DROP TABLE goods;
#② 还原数据
mysql> source ~/mydb.sql
#③ 查看数据是否还原
mysql> SELECT id, name FROM goods;
+-------+-------+
| id    | name  |
+-------+-------+
|   1   | book  |
|   2   | apple |
+-------+-------+
2 rows in set (0.00 sec)
```

从查询结果可以看出，数据已经正确还原。

12.3.3　二进制日志

二进制日志用于记录 MySQL 数据库的变化，如表的创建、对表数据的更改，记录涉及修改的 SQL、数据修改的行变化、执行时间等信息。二进制日志可以应用于搭建复制架构（如主从复制）、数据恢复等场景。下面介绍二进制日志的使用。

1. 开启二进制日志

默认情况下，二进制日志是关闭的，在 my.cnf 配置文件的“[mysqld]”区段中增加如下配置可以开启二进制日志。

```
log-bin=binlog
server-id=1
```

在上述配置中，log-bin 用于开启二进制日志，binlog 表示二进制日志的文件名。在开启二进制日志时需要使用 server-id 指定服务器 id，用于在复制架构中区分服务器，确保每个服务器的 id 不同即可。值得一提的是，log-bin 后面的“=文件名”可以省略，省略后会自动使用服务器主机名作为文件名，如 localhost。

保存配置后，执行如下命令重新启动 MySQL 服务，配置就会生效。

```
[root@localhost ~]# systemctl restart mysqld
```

生效以后，登录 MySQL，执行如下语句可以查看二进制日志的开启情况。

```
mysql> SHOW VARIABLES LIKE 'log_bin%';
+-----------------------------------+------------------------------+
| Variable_name                     | Value                        |
+-----------------------------------+------------------------------+
| log_bin                           | ON                           |
| log_bin_basename                  | /var/lib/mysql/binlog        |
| log_bin_index                     | /var/lib/mysql/binlog.index  |
| log_bin_trust_function_creators   | OFF                          |
| log_bin_use_v1_row_events         | OFF                          |
+-----------------------------------+------------------------------+
5 rows in set (0.00 sec)
```

从上述信息可以看出，二进制日志已经开启，日志文件的保存目录为/var/lib/mysql，基本文件名(basename)为 binlog，索引文件为 binlog.index。

log_bin_trust_function_creators 表示是否可以信任函数创建者，由于在复制架构中，函数有可能导致主从的数据不一致，MySQL 会限制函数的创建、修改和调用。

log_bin_use_v1_row_events 表示是否使用版本 1 的二进制日志行事件，默认为 OFF，表示使用版本 2 的二进制日志行事件。

2. 查看二进制日志文件

在/var/lib/mysql 目录中查看二进制日志文件，具体命令和结果如下。

```
[root@localhost ~]# ls /var/lib/mysql | grep binlog
binlog.000001
binlog.index
```

从上述结果可以看出，当前二进制日志有两个文件，扩展名分别是 000001 和 index。index 是索引文件，使用 cat 命令查看该文件，结果如下。

```
[root@localhost ~]# cat /var/lib/mysql/binlog.index
./binlog.000001
```

binlog.000001 文件保存二进制日志的内容，当 MySQL 服务重新启动，或文件大小达到了 max_binlog_size 配置的上限(默认为 1GB)，就会创建一个新日志文件，新日志文件的扩展名加 1 递增，即 binlog.000002。

在登录 MySQL 后，使用 SHOW BINARY LOGS 语句也可以查看二进制日志文件，具体结果如下。

```
mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000001 |       154 |
+---------------+-----------+
1 row in set (0.00 sec)
```

在上述结果中，Log_name 表示日志文件名，File_size 表示文件大小(字节)。

3. 查看二进制日志内容

二进制日志采用二进制方式保存，无法直接查看文件内容，需要使用 mysqlbinlog 工具将其转换为文本格式的 SQL 脚本，具体命令和结果如下。

```
[root@localhost ~]# cd /usr/local/mysql/bin
[root@localhost bin]# ./mysqlbinlog /var/lib/mysql/binlog.000001
/* !50530 SET @@SESSION.PSEUDO_SLAVE_MODE=1 */;
/* !50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0 */;
DELIMITER /* ! */;
ROLLBACK/* ! */;
BINLOG '
u1VSWw8BAAAAdwAAAHsAAAABAAQANS43LjIyLWxvZwAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAACgoKKioAEjQAAaQ8NVo=
'/* ! */;
# at 123
# 180723  5:35:55 server id 1  end_log_pos 154 CRC32 0x7e392ba9
  ⋮(此处省略一些显示结果)
```

在上述信息中，“at 123”记录了日志的位置，180723 5：35：55 记录了日志的时间。

将二进制日志转换为 SQL 脚本后，可以直接导入到 MySQL 中，具体命令如下。

```
[root@localhost bin]# ./mysqlbinlog /var/lib/mysql/binlog.000001 | mysql -uroot -p123456
```

在上述命令中，管道符号“|”可以在 Linux 或 Windows 系统中使用，它用于将左边的 mysqldump 命令的输出结果作为右边的 mysql 命令的输入，实现数据导入。

利用二进制日志可以灵活地进行数据备份与恢复，尤其是当数据库中的数据量非常大时，如果采用完整备份的方式，每次备份的数据量会非常大，速度也非常慢。而二进制日志记录了数据库发生过的变化，定期备份二进制日志文件就可以实现增量备份的效果。

在使用 mysqlbinlog 命令时，还可以添加一些选项，具体语法如下。

```
mysqlbinlog [options] log-files
```

在上述语法中，log-files 是日志文件路径，options 是可选的选项。通过选项可以指定根据日期、位置等来恢复指定区间的数据变化，具体如下。

- --start-date 和--stop-date：指定开始日期和结束时期。
- --start-position 和--stop-position：指定开始位置和结束位置。

下面以恢复到 2018-07-23 10:12:20 之前所有的操作为例进行演示，具体命令如下。

```
[root@localhost bin]# ./mysqlbinlog --stop-date='2018-07-23 10:12:20' /var/lib/mysql/binlog.000001 |
mysql -uroot -p123456
```

若要恢复到指定位置，可以在输出的日志结果中通过“at 位置”来查看，也可以使用 SHOW MASTER STATUS 查看当前位置，具体如下。

```
mysql> SHOW MASTER STATUS\G
*************************** 1. row ***************************
```

```
            File: binlog.000002
        Position: 154
    Binlog_Do_DB:
Binlog_Ignore_DB:
Executed_Gtid_Set:
1 row in set (0.00 sec)
```

在上述信息中,“File: binlog.000002”表示当前所在的日志文件,“Position: 154”表示当前所在的位置。

4. 暂停二进制日志

如果需要临时暂停记录二进制日志,可以用如下语句实现。

```
mysql>SET sql_log_bin=0;
```

上述操作只针对当前会话有效,当退出 MySQL 客户端后,下次登录就会恢复记录。如果需要立即恢复二进制日志的记录,可以用如下语句实现。

```
mysql>SET sql_log_bin=1;
```

上述语句执行后,二进制日志就会继续开始记录。

5. 删除二进制日志

在 MySQL 服务器运行的情况下,若直接删除日志文件,可能会导致程序出错。推荐使用 RESET 或 PURGE 语句进行删除,具体示例如下。

```
#方式 1 删除所有二进制日志文件
mysql>RESET MASTER;
#方式 2 删除创建时间早于 binlog.000002 的所有日志文件
mysql>PURGE MASTER LOGS TO 'binlog.000002';
#方式 3 删除 2018 年 7 月 23 日前的二进制日志
mysql>PURGE MASTER LOGS BEFORE '20180723';
```

在上述命令中,第 2 种方式删除的日志文件不包括 binlog.000002 文件本身,第 3 种方式删除的日志不包括 2018 年 7 月 23 日当天的日志。删除二进制日志后,读者可通过 SHOW BINARY LOGS 语句查看删除结果。

12.4 多实例部署

MySQL 支持多实例部署,多实例是指在一台服务器中运行多个 MySQL 服务,通过监听不同的端口号来区分。当一台服务器的资源有剩余时,通过多实例部署,可以利用这些资源来提供更多服务。下面将演示如何实现 MySQL 多实例部署。

(1) 更改配置文件。MySQL 多实例部署是通过更改配置文件实现的,执行如下命令开始编辑。

```
[root@localhost ~]# vi /etc/my.cnf
```

在配置文件中增加如下配置。其中，“[mysqld@replica01]”和“[mysqld@replica02]”区段分别表示两个实例，每个实例拥有自己的 port、datadir、socket 和 log-error 配置。

```
[mysqld@replica01]
port=3307
datadir=/var/lib/mysql-replica01
socket=/var/lib/mysql-replica01/mysql.sock
log-error=/var/lib/mysql-replica01/mysqld.log
[mysqld@replica02]
port=3308
datadir=/var/lib/mysql-replica02
socket=/var/lib/mysql-replica02/mysql.sock
log-error=/var/lib/mysql-replica02/mysqld.log
```

添加上述配置后，保存配置文件。此时 MySQL 一共有 3 个实例，第 1 个实例就是前面一直使用的监听 3306 端口的 MySQL 服务，第 2 个和第 3 个实例则是通过上述配置增加的，分别监听 3307 和 3308 端口。

（2）初始化数据库。使用 mysqld 命令为新增的两个实例初始化数据库，具体命令如下。

```
[root@localhost ~]# cd /usr/local/mysql/bin
[root@localhost bin]# ./mysqld --initialize-insecure --datadir=/var/lib/mysql-replica01
[root@localhost bin]# ./mysqld --initialize-insecure --datadir=/var/lib/mysql-replica02
```

在上述命令中，mysqld 的选项--datadir 指向了配置文件中为两个新实例配置的 datadir 目录。初始化后，这两个实例的 root 用户的密码都为空。

（3）启动多实例服务。在支持 systemd 的 CentOS7 系统中，需要使用 mysqld@.service 脚本来管理多实例服务。对于 yum 安装方式，已经自动安装了该脚本，而对于编译安装方式，需要手动复制脚本文件到系统目录，具体操作如下。

```
#① 切换到 MySQL 源代码目录中的 scripts 目录
[root@localhost bin]# cd ~/mysql-5.7.22/scripts
#② 复制脚本文件
[root@localhost scripts]# cp mysqld@.service /usr/lib/systemd/system/
```

接下来就可以使用如下命令启动多实例服务，并设置开机启动。

```
#① 启动实例 mysqld@replica01 和 mysqld@replica02
[root@localhost bin]# systemctl start mysqld@replica01
[root@localhost bin]# systemctl start mysqld@replica02
#② 设置开机自动启动
[root@localhost bin]# systemctl enable mysqld@replica01
[root@localhost bin]# systemctl enable mysqld@replica02
```

为了确认多实例是否已经启动，执行如下命令查看监听的端口号。

```
[root@localhost bin]# ss -tnlp | grep mysql
LISTEN   0   80   :::3306    :::*    users:(("mysqld",pid=2709,fd=21))
LISTEN   0   80   :::3307    :::*    users:(("mysqld",pid=2820,fd=21))
LISTEN   0   80   :::3308    :::*    users:(("mysqld",pid=10534,fd=21))
```

从上述结果可以看出，mysqld 监控了 3306、3307 和 3308 端口，说明 MySQL 的 3 个实例都已经启动。

(4) 登录并设置密码。使用 MySQL 客户端工具分别登录新创建的两个实例，并在登录后设置 root 用户的密码为 123456，具体命令如下。

```
#① 登录 3307 端口的 mysqld@replica01 实例
[root@localhost bin]# mysql -h127.0.0.1 -P3307 -uroot
#② 设置密码
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY '123456';
mysql> exit
#③ 登录 3308 端口的 mysqld@replica02 实例
[root@localhost bin]# mysql -h127.0.0.1 -P3308 -uroot
#④ 设置密码
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY '123456';
mysql> exit
```

在上述命令中，mysql 命令的选项-P 表示端口号。在设置密码后，下次登录时需要加上选项"-p123456"。

小提示：在后面的小节中，主从复制和组复制都需要用到多实例环境，但是这两种方案是互相冲突的。为了更方便地进行操作，建议读者在完成多实例环境的搭建后，使用 VMware 拍摄一个虚拟机快照，在完成主从复制后，还原快照，再进行组复制的搭建。

12.5　主从复制

MySQL 支持数据库复制(Replication)功能，从一台 MySQL 主服务器(Master)将数据复制到另一台或多台 MySQL 从服务器(Salves)，这个过程就是主从复制。主从复制是通过二进制日志实现的，主服务器的二进制日志将会发送给从服务器执行，从而使从服务器的数据与主服务器同步。主从复制的优点是从服务器可以分担主服务器的压力，并且从服务器可以作为主服务器的备份，防止因其中一台服务器故障导致数据丢失。

下面将演示如何实现 MySQL 的主从复制。由于搭建主从复制环境至少需要两台服务器，部署比较麻烦，为了简化操作，可以在前面搭建的多实例环境中实现主从复制，将实例 mysqld@replica01 作为主服务器，实例 mysqld@replica02 作为从服务器。

(1) 更改配置文件。使用 vi 编辑器打开 MySQL 配置文件，具体命令如下。

```
[root@localhost ~]# vi /etc/my.cnf
```

在配置文件中的"[mysqld@replica01]"区段中开启二进制日志，并配置服务器 id。

```
log-bin=binlog
server-id=2
```

然后在“[mysqld@replica02]”区段中配置从服务器的 id。

```
server-id=3
```

配置完成后，重新启动这两个实例的 MySQL 服务，使配置生效。

```
[root@localhost ~]# systemctl restart mysqld@replica01
[root@localhost ~]# systemctl restart mysqld@replica02
```

（2）在主服务器中创建用户。主服务器 mysqld@replica01 的端口号为 3307，执行如下命令登录主服务器。

```
[root@localhost ~]# mysql -h127.0.0.1 -P3307 -uroot -p123456
```

登录主服务器后，创建用于从服务器访问的用户 slave，指定密码为 123456。

```
# ① 创建 slave 用户
mysql> CREATE USER 'slave'@'127.0.0.1' IDENTIFIED BY '123456';
# ② 为用户赋予 REPLICATION SLAVE 权限
mysql> GRANT REPLICATION SLAVE ON *.* TO 'slave'@'127.0.0.1';
```

由于这里使用一台服务器实现主从复制，因此 slave 用户的主机地址为 127.0.0.1。若为两台独立的服务器进行主从复制部署，则 slave 用户的主机地址是从服务器的 IP 地址，并且需要在主服务器的防火墙中开放 3306 端口。

（3）查看主服务器当前的二进制日志状态。执行如下命令查看主服务器的二进制日志状态，这些信息将在从服务器中使用。

```
mysql> SHOW MASTER STATUS;
+---------------+----------+--------------+------------------+-------------------+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+---------------+----------+--------------+------------------+-------------------+
| binlog.000001 |      623 |              |                  |                   |
+---------------+----------+--------------+------------------+-------------------+
1 row in set (0.00 sec)
```

从上述信息可以看出，当前日志文件为 binlog.000001，位置为 623。

完成上述操作后，执行 exit 命令退出主服务器。

```
mysql> exit
```

（4）从服务器同步数据。从服务器 mysqld@replica02 的端口号为 3308，执行如下命令登录从服务器。

```
[root@localhost ~]# mysql -h127.0.0.1 -P3308 -uroot -p123456
```

登录后，执行如下 SQL 语句实现从服务器自动同步主服务器。

```
# ① 指定主服务器的信息
mysql> CHANGE MASTER TO MASTER_HOST='127.0.0.1', MASTER_PORT=3307,
```

```
    ->MASTER_LOG_FILE='binlog.000001', MASTER_LOG_POS=623;
#② 使用 slave 用户开始同步
mysql>START SLAVE USER='slave' PASSWORD='123456';
```

在上述命令中，CHANGE MASTER TO 用于更改从服务器连接主服务器的参数，MASTER_HOST 表示主服务器的主机，MASTER_PORT 表示主服务器的端口号，MASTER_LOG_FILE 表示主服务器的二进制日志文件，MASTER_LOG_POS 表示主服务器的位置。USER 和 PASSWORD 表示访问主服务器的用户名和密码。

（5）查看从服务器的同步状态。执行 SHOW SLAVE STATUS 语句查看从服务器的同步状态，如下所示。

```
mysql>SHOW SLAVE STATUS\G
*************************** 1. row ***************************
              Slave_IO_State: Waiting for master to send event
                 Master_Host: 127.0.0.1                      #主服务器的主机
                 Master_User: slave                          #主服务器的用户名
                 Master_Port: 3307                           #主服务器的端口号
               Connect_Retry: 60                             #连接失败重试秒数
             Master_Log_File: binlog.000001                  #主服务器二进制日志文件
         Read_Master_Log_Pos: 623                            #主服务器的日志读取位置
              Relay_Log_File: localhost-relay-bin.000002     #从服务器的中继日志文件
               Relay_Log_Pos: 319                            #从服务器中继日志的位置
       Relay_Master_Log_File: binlog.000001                  #主服务器的中继日志文件
            Slave_IO_Running: Yes                            #从服务器 IO 线程是否运行
           Slave_SQL_Running: Yes                            #从服务器 SQL 线程是否运行
     ⋮（此处省略一些显示结果）
1 row in set (0.00 sec)
```

在上述信息中，Slave_IO_Running 表示是否连接到主服务器，从服务器读取主服务器的二进制日志到本地，保存为中继日志（Relay Log）；Slave_SQL_Running 表示是否可以读取中继日志，并执行日志中的 SQL 语句。Slave_IO_State（从服务器的 IO 状态）为 Waiting for master to send event，表示正在等待主服务器发送事件，当主服务器中的二进制日志发生变化时，就会利用事件来通知从服务器同步数据。

完成上述操作后，执行 exit 命令退出从服务器。

```
mysql>exit
```

（6）主从复制测试。在主服务器中写入数据，测试从服务器是否同步，具体操作如下。

```
#① 登录主服务器
[root@localhost ~]#mysql -h127.0.0.1 -P3307 -uroot -p123456
#② 创建 mydb 数据库和 test 数据表，插入测试数据
mysql>CREATE DATABASE mydb;
mysql>CREATE TABLE mydb.test (id INT);
mysql>INSERT INTO mydb.test VALUES(1);
#③ 退出主服务器
mysql>exit
#④ 登录从服务器
```

```
[root@localhost ~]#mysql -h127.0.0.1 -P3308 -uroot -p123456
#⑤ 在从服务器中读取数据
mysql>SELECT * FROM mydb.test \G
*************************** 1. row ***************************
id: 1
1 row in set (0.00 sec)
```

从上述操作可以看出，在主服务器中写入数据后，从服务器中也具有了相同的数据，说明从服务器可以同步主服务器的数据。

接下来测试在从服务器中写入数据，主服务器将不会同步，具体操作如下。

```
#① 在从服务器中写入数据
mysql>INSERT INTO mydb.test VALUES(2);
#② 退出从服务器
mysql>exit
#③ 登录主服务器
[root@localhost ~]#mysql -h127.0.0.1 -P3307 -uroot -p123456
#④ 在主服务器中读取数据
mysql>SELECT * FROM mydb.test\G
*************************** 1. row ***************************
id: 1
1 row in set (0.00 sec)
```

从上述结果可以看出，从服务器写入数据后，主服务器中并没有同步。因此，在实际使用时，从服务器负责查询操作，而对于添加、修改、删除操作则在主服务器中进行。

12.6 动手实践：组复制

数据库的学习在于多看、多学、多思考、多动手，只有将理论与实际相结合，才能够体会数据库开发与管理的重要性，展现知识学习的价值与力量。接下来请结合本章所学的知识完成 MySQL 组复制环境的搭建。

【实践目标】

本实践的目标是能够根据文字提示，完成 MySQL 组复制环境的搭建，理解 MySQL 组复制架构的作用和优势。

【实践需求】

MySQL 组复制(MySQL Group Replication)是 MySQL 推出的一个高可用、易于扩展、可靠性强的部署方案，它具有良好的扩展能力，可以动态增加或删除成员，确保组内数据的一致性。需要注意的是，MySQL 组复制功能要求使用事务存储引擎 InnoDB。

MySQL 组复制支持单主模式和多主模式，本实践将会搭建单主模式，其结构如图 12-8 所示。

在图 12-8 中，服务器 1 是主节点，可读可写，服务器 2 和服务器 3 是从节点，它们的数据与主节点是同步的，并且只允许读取数据。

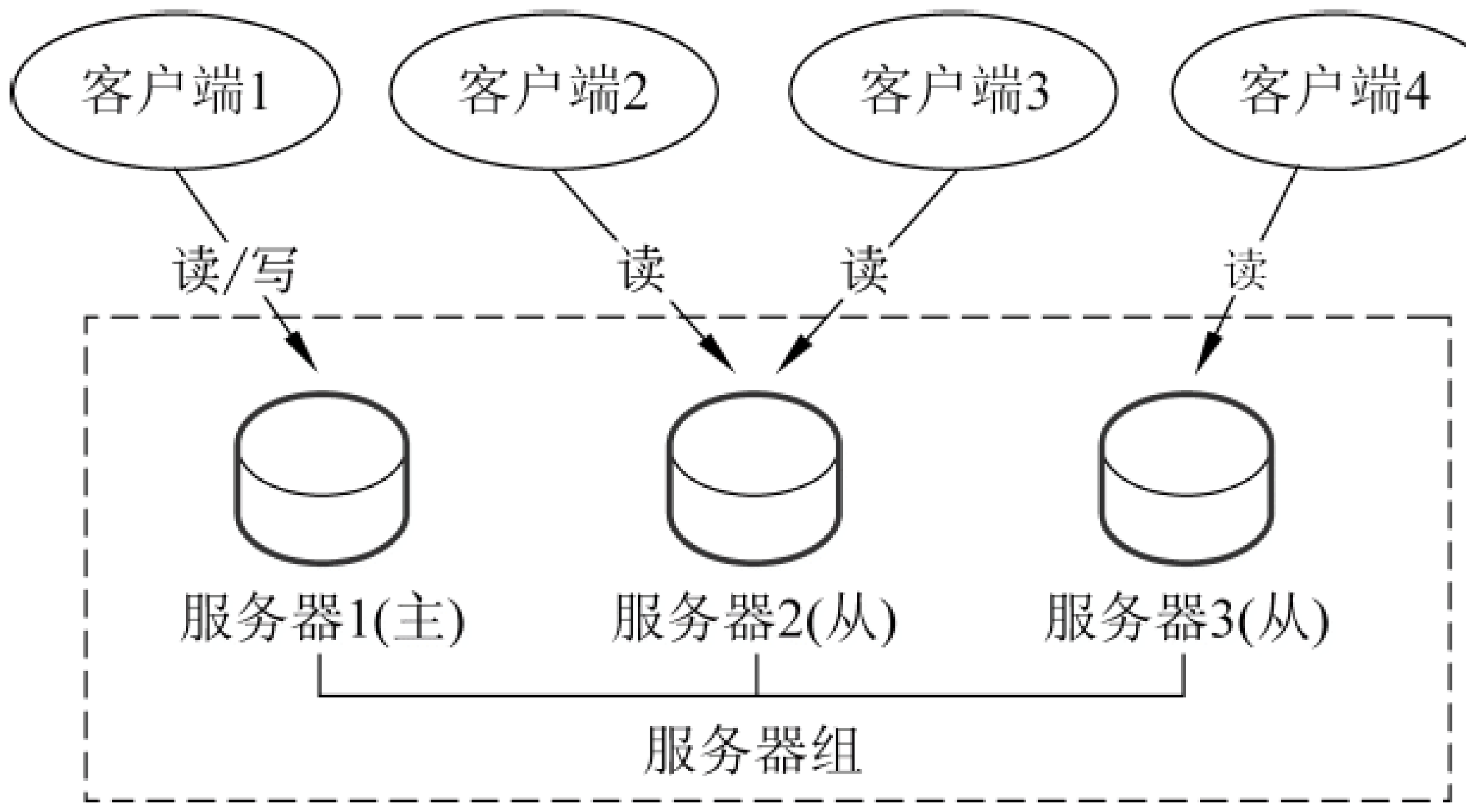


图 12-8 MySQL 组复制单主模式

需要注意的是，在组复制中，主节点并不是固定的，若主节点发生宕机，会自动选举一个从节点替代主节点，允许写入数据，并和其他从节点数据同步。

小提示：由于组复制和主从复制环境是冲突的，读者在搭建组复制前，应将虚拟机快照还原到 12.4 小节多实例部署完成时的状态。

【动手实践】

1. 更改配置文件

在 MySQL 配置文件中进行组复制的配置，执行如下命令编辑配置文件。

```
[root@localhost ~]# vi /etc/my.cnf
```

在“[mysqld]”区段中添加复制功能的配置，具体如下。

```
server_id=1                              #服务器的 id,用于区分服务器
gtid_mode=ON                             #开启 GTID(全局事务 ID)模式
enforce_gtid_consistency=ON              #保证 GTID 的安全
master_info_repository=TABLE             #主服务器信息记录为表
relay_log_info_repository=TABLE          #中继日志信息记录为表
binlog_checksum=NONE                     #二进制日志不进行校验
log_slave_updates=ON                     #接收的更新是否记录到自己的二进制日志中
log_bin=binlog                           #开启二进制日志
binlog_format=ROW                        #设置二进制日志格式为 ROW
```

在上述配置中，“#”注释用于介绍这些配置的作用，无须在配置文件中编写。其中，server_id 和 log_bin 在前面已经讲过，其他配置也是组复制所需的基本配置。

接下来在上述配置的下面添加组复制的配置，具体如下。

```
#记录事务的算法,此处使用 64 位的 xxHash 标识来记录事务
transaction_write_set_extraction=XXHASH64
#配置组名称,需要使用 UUID 格式
loose-group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
#是否随服务器启动而自动启动组复制
loose-group_replication_start_on_boot=OFF
```

```
#本地的 IP 地址和端口号
loose-group_replication_local_address="127.0.0.1:33060"
#组中的各成员的 IP 地址和端口号
loose-group_replication_group_seeds="127.0.0.1:33060,127.0.0.1:33061,127.0.0.1:33062"
#是否是组中的引导节点
loose-group_replication_bootstrap_group=OFF
```

在上述配置中，组名称可以随意填写，但要确保同一个组内的服务器的名称一致。组名称为 UUID 格式，在 CentOS 中使用 uuidgen 命令可以生成一个 UUID，如下所示。

```
[root@localhost ~]#uuidgen
00cc79b2-0ecd-4382-b994-f25e5e1db2ea
```

在 loose-group_replication_group_seeds 中配置了 3 个成员，IP 地址都是 127.0.0.1，端口号分别是 33060、33061 和 33062，对应 mysqld、mysqld@replica01 和 mysqld@replica02 实例。需要注意的是，这些端口号并不是 MySQL 服务的端口号(3306、3307 和 3308)，在组复制环境中，成员服务器之间会使用专门的 IP 地址和端口号进行通信。因此，在 loose-group_replication_local_address 配置中使用 33060 作为端口号。

在完成"[mysqld]"区段的配置之后，组中的第一个成员就配置完成了，另外两个成员的配置与第一个成员是大致相同的，可以将上述为"[mysqld]"区段添加的配置复制给另外两个区段"[mysqld@replica01]""[mysqld@replica02]"使用。复制完成后，再修改这两个区段中的两处不同的配置，具体如下。

```
#"[mysqld@replica01]"区段的不同配置
server_id=2
loose-group_replication_local_address="127.0.0.1:33061"
#"[mysqld@replica02]"区段的不同配置
server_id=3
loose-group_replication_local_address="127.0.0.1:33062"
```

修改完成后，保存配置文件，然后重新启动这 3 个服务实例使配置生效。

```
[root@localhost ~]#systemctl restart mysqld
[root@localhost ~]#systemctl restart mysqld@replica01
[root@localhost ~]#systemctl restart mysqld@replica02
```

2. 创建用户并指定复制通道

组复制与主从复制相似，都需要创建用于其他服务器访问的用户。其区别是，组复制的每台服务器都需要创建用户，因为每一台服务器都有可能成为主节点。

接下来通过如下命令分别登录 3 个 MySQL 实例，然后执行相同的操作。

```
[root@localhost ~]#mysql -uroot -p123456
[root@localhost ~]#mysql -h127.0.0.1 -P3307 -uroot -p123456
[root@localhost ~]#mysql -h127.0.0.1 -P3308 -uroot -p123456
```

在每个 MySQL 实例中执行的相同操作如下所示。

```
#① 创建 rpl 用户,赋予 REPLICATION SLAVE 权限
mysql> SET SQL_LOG_BIN=0;
mysql> CREATE USER rpl@'%' IDENTIFIED BY '123456';
mysql> GRANT REPLICATION SLAVE ON *.* TO rpl@'%';
mysql> SET SQL_LOG_BIN=1;
#② 指定通道为 group_replication_recovery
mysql> CHANGE MASTER TO MASTER_USER='rpl', MASTER_PASSWORD='123456'
    -> FOR CHANNEL 'group_replication_recovery';
#③ 安装组复制插件(搭建组复制需要依赖此插件)
mysql> INSTALL PLUGIN group_replication SONAME 'group_replication.so';
```

在上述操作中,创建用户前后的 SET SQL_LOG_BIN 用于暂停二进制日志,避免将创建用户的操作记录在二进制日志中。CHANGE MASTER 语句与主从复制的区别在于,无须指定主服务器,而是指定复制通道为 group_replication_recovery(组复制恢复)。在组复制中,如果一个新的成员加入到组中,它需要将自己的数据和组内的数据保持同步,也就是与提供二进制日志的成员服务器建立 group_replication_recovery 通道来恢复数据。

3. 启动组复制

安装组复制插件后,就可以启动组复制了。在一个新创建的组中,需要有一个成员是引导节点,负责创建组,创建之后其他成员才可以加入组。具体操作如下。

```
#① 登录第 1 个实例
[root@localhost ~]# mysql -uroot -p123456
#② 引导节点创建组
mysql> SET GLOBAL group_replication_bootstrap_group=ON;
mysql> START GROUP_REPLICATION;
mysql> SET GLOBAL group_replication_bootstrap_group=OFF;
```

在创建组以后,第一个加入组的成员将被选为主节点。使用如下 SQL 语句可以查看当前组内的成员信息,当前组内只有一个成员,即主节点。

```
mysql> SELECT * FROM performance_schema.replication_group_members\G
*************************** 1. row ***************************
CHANNEL_NAME: group_replication_applier              #通道名称
   MEMBER_ID: 341d0ec2-8b33-11e8-a0b9-000c29f72b59   #成员 id
 MEMBER_HOST: localhost.localdomain                  #成员主机
 MEMBER_PORT: 3306                                   #成员端口号
MEMBER_STATE: ONLINE                                 #成员状态
1 row in set (0.02 sec)
```

在上述信息中,若 MEMBER_STATE 的值为 ONLINE,表示该成员是有效的节点,可以进行组内通信,可以对外提供服务。

将另外两个实例加入组,执行 START GROUP_REPLICATION 即可,如下所示。

```
#① 第 2 个实例加入组
[root@localhost ~]# mysql -h127.0.0.1 -P3307 -uroot  -p123456
mysql> START GROUP_REPLICATION;
```

```
#② 第3个实例加入组
[root@localhost ~]#mysql -h127.0.0.1 -P3308 -uroot  -p123456
mysql>START GROUP_REPLICATION;
```

当组内有成员加入以后，再次查看组内成员，如下所示。

```
mysql>SELECT MEMBER_HOST, MEMBER_PORT, MEMBER_STATE FROM
    ->performance_schema.replication_group_members;
+-----------------------+-------------+--------------+
| MEMBER_HOST           | MEMBER_PORT | MEMBER_STATE |
+-----------------------+-------------+--------------+
| localhost.localdomain |        3306 | ONLINE       |
| localhost.localdomain |        3308 | ONLINE       |
| localhost.localdomain |        3307 | ONLINE       |
------------------------+-------------+--------------+
3 rows in set (0.00 sec)
```

从上述结果可以看出，组内已经有了3个成员，且都是有效节点。

需要注意的是，组复制环境中只有主节点允许写入，如果在从节点中执行写操作，则会报错，提示当前处于只读模式，无法执行该语句，如下所示。

```
mysql>CREATE DATABASE newdb;
ERROR 1290 (HY000): The MySQL server is running with the --super-read-only option so it cannot execute
this statement
```

4. 主节点宕机测试

为了测试组复制环境中，主节点宕机后是否会重新选举一个主节点，将主节点对应的第1个实例停止，如下所示。

```
[root@localhost ~]#systemctl stop mysqld
```

接下来登录到组成员列表中的第2个节点，即3308端口的实例，如下所示。

```
[root@localhost ~]#mysql -h127.0.0.1 -P3308 -uroot -p123456
```

登录以后，创建数据库进行测试。

```
mysql>CREATE DATABASE newdb;
Query OK, 1 row affected (0.01 sec)
```

上述语句执行以后，在3307实例中查看数据是否同步，如下所示。

```
[root@localhost ~]#mysql -h127.0.0.1 -P3307 -uroot -p123456
mysql>SHOW DATABASES LIKE 'newdb'\G
*************************** 1. row ***************************
Database (newdb): newdb
1 row in set (0.01 sec)
```

从以上测试可以看出，当主节点停止后，3308 端口的实例自动变成了主节点，可以写入数据了，并且写入后在 3307 端口的实例中也可以看到同步的结果。

接下来启动第 1 个实例，通过 START GROUP_REPLICATION 语句重新加入到组，然后查看数据是否已经同步，如下所示。

```
#① 启动第 1 个实例,然后登录 MySQL
[root@localhost ~]#systemctl start mysqld
[root@localhost ~]#mysql -uroot -p123456
#② 在加入组前,查看是否存 newdb 数据库
mysql>SHOW DATABASES LIKE 'newdb';
Empty set (0.00 sec)
#③ 加入到组
mysql>START GROUP_REPLICATION;
#④ 再次查看是否存在 newdb 数据库
mysql>SHOW DATABASES LIKE 'newdb'\G
*************************** 1. row ***************************
Database (newdb): newdb
1 row in set (0.01 sec)
```

从上述结果可以看出，由于 newdb 数据库是在第 1 个实例停止期间创建的，因此在恢复第 1 个实例后，还没有 newdb 数据库。在将第 1 个实例加入组后，就会利用组中最新的二进制日志来恢复数据，因此就可以查询到 newdb 数据库。

12.7　本章小结

本章主要讲解了如何在 Linux 系统中安装、配置和部署 MySQL，以及如何进行数据备份与还原，如何利用主从复制或组复制来提高数据库的可用性和负载能力。读者应重点掌握 MySQL 的安装和配置方法、数据全量和增量备份的实现，以及二进制日志的使用。

12.8　课后练习

一、填空题

1. 在 CentOS 7.5 中启动 MySQL 服务的命令是________。
2. MySQL 配置文件中关于服务器的配置写在________区段中。
3. MySQL 服务器默认字符集的配置是________。
4. MySQL 允许客户端最大同时连接数的配置是________。
5. MySQL 开启二进制日志的配置是________。

二、判断题

1. Windows 和 Linux 系统中数据库名都不区分大小写。(　　)
2. MySQL 配置文件的路径是/etc/mysql.cnf。(　　)
3. 查看二进制日志使用 mysqlbinlog 命令。(　　)

4. 使用 mysqld --initialize 初始化数据库后 root 用户的密码为空。(　　)
5. 开启二进制日志后，执行 FLUSH LOG 会生成新的日志文件。(　　)

三、选择题

1. 在 Linux 系统中，远程下载的命令是(　　)。
   A. download　　B. get　　C. wget　　D. load
2. 在 Linux 系统中，表示输入重定向的符号是(　　)。
   A. |　　B. ＜　　C. ＞　　D. -
3. 查看二进制日志状态使用(　　)语句。
   A. PURGE BINARY LOGS　　B. SHOW BINARY STATUS
   C. SHOW MASTER STATUS　　D. SHOW MASTER LOGS
4. 指定二进制日志缓存大小的配置是(　　)。
   A. binary_cache_size　　B. binlog_cache_size
   C. max_binlog_size　　D. binlog_max_size
5. 在遇到(　　)情况下，二进制日志文件会递增。
   A. MySQL 服务重新启动
   B. 达到 max_binlog_size 的上限
   C. 执行 FLUSH LOG 语句
   D. 执行 FLUSH PRIVILEGES 语句

四、简答题

1. 请简述常规日志和二进制日志的区别。
2. 请简述备份 MySQL 数据库的方案及优缺点。

五、实训题

1. 请动手实现，利用二进制日志备份和恢复数据。
2. 请动手实现，将 sh_goods 表中的数据导出成 XML 格式。